Agents-oriented platform to discriminate activity patterns

Oscar Franco Genis

August 10, 2009
Contents

1 Introduction 6
  1.1 Introduction ............................................. 6
  1.2 State of art ............................................. 8

2 Project goals 12
  2.1 Capture .................................................. 13
  2.2 Pre-Processing .......................................... 13
  2.3 Diagnose ............................................... 14
  2.4 Action .................................................... 14

3 Specification 16
  3.1 Hardware ............................................... 16
    3.1.1 Mainboard ........................................... 16
    3.1.2 Sensors ............................................... 17
  3.2 Software ............................................... 20
    3.2.1 Structure ............................................. 20
    3.2.2 Class diagram ....................................... 21
4 Activity pattern detection

4.1 Signal capturing ................................................. 22
4.2 Frequency treatment ........................................... 25
4.3 Classification .................................................... 27

5 Design .................................................................. 31

5.1 Jade .................................................................. 32
5.2 C/C++ .................................................................. 37
5.3 Qt ..................................................................... 39

6 Implementation ......................................................... 40

6.1 Capture ............................................................... 40
   6.1.1 Sensors server application .............................. 40
   6.1.2 CaptureConSensor.java ................................. 44
   6.1.3 Capture Agent ............................................. 45

6.2 Previous processing treatment ............................... 51
   6.2.1 Pre-Process Agent ....................................... 51

6.3 Diagnose ............................................................ 55
   6.3.1 Classifier Agent .......................................... 55

6.4 Actuator ............................................................. 58
   6.4.1 Gui: LCD Agent ......................................... 59
   6.4.2 LCDConQt.java ......................................... 61
   6.4.3 Qt Application ........................................... 62

6.5 Demonstration Testing application ....................... 63

6.6 Compatibility withdrawals ................................... 66

6.7 Compilation ........................................................ 70
6.7.1 Java .................................................. 70
6.7.2 C++ .................................................. 71
6.7.3 Qt .................................................... 71

7 Conclusions ........................................... 72

7.1 Future Considerations and improvements ............. 73
List of Figures

1.1 Activity pattern monitoring ............................... 7
1.2 Heart Signal ........................................... 7
1.3 Daphnet ............................................... 8
1.4 Nano Tera ............................................. 9
1.5 VCU - USA ........................................... 10
1.6 CSEM: Wearable sensors ............................... 10
1.7 CSEM System ......................................... 11
1.8 Wearable System Proposal by University of Alabama 11

2.1 Main Structure ......................................... 13
2.2 Possible Actuators ..................................... 14

3.1 Ubidule Perplexus ..................................... 17
3.2 Toradex Sensors ........................................ 18
3.3 Tilt Spherical coordinates ............................ 20
3.4 Structure ............................................... 21

4.1 Accelerometer: 256 Values X,Y & Z Axis ............... 23
4.2 512 Values of Channel Y-axis in time ................. 24
4.3 FFT: time to frequency domain ....................... 25
Chapter 1

Introduction

1.1 Introduction

People who needs attention or to be under supervision, including elders or hospitalized people, could be considered subjects of study. They are involved in quotidian situations where it would be interesting to keep informed about their health status.

The development of a wearable health system could be the solution in order to extend services to health care, for instance, in geriatric centers. Also it can be interesting to monitor health condition and sport performance, for real time prevention of injuries in professional players during training.

One possible solution to perform a custom diagnostic about someone’s status who is under supervision, Figure 1.1, can be to keep up-to-date about his/her daily activities or behaviours.

Most of these activities will respond to repetitive patterns. This periodicity is observable in human vital signs measurement, Figure 1.2, like cardiovascular and respiration rate, as well as in the activities related with the movement
like walking or running, etc... It would be also interesting, for example, to detect anomalous situations such as falling down or fainting.

Detection of body movement, measurement of vital signs as electrocardiogram, heart rate and oxygen saturation are easily obtainable through the appropriate set of sensors attached to the body. With all this information we can establish detection models for each type of activity or behaviour and deduce the status of the person who wear the sensors.

This project starts with a long learning & research process. Firstly, getting knowledge about Perplexus (acronyms for Pervasive computing framework for modelling complex virtually unbounded systems): distributed programming, ontologies, Jade. And then, looking about health systems researches, undergoing studies and sensors involved in this kind of systems.
1.2 State of art

Developing a wearable health system to extend services for health care is an objective that is expanding in two different directions: sensors (develop tissues with sensing capacity, improving connectivity of sensors...), development of signal processing techniques in order to analyze the vital signals emitted by the user’s body and also developing decision methods based on ubiquitous connectivity.

Daphnet[4] project at ETH Zurich, researches on the properties of different sensor and their signals, see Figure 1.3. The physiological functions studied, are the cardio vascular capacities, brain activity (EEG), motor control, gait, posture, sleep and sympathetic and parasympathetic effects. Algorithms to assess the long-term effects and interrelations between signals representing these functions are at the core of that project.

“The simultaneous collection of several long-term signals will enable the construction of physiological networks using dynamical synchronisation and cross-correlations patterns, whose momentary state together with the properties of each signal can give a picture of the health status of an individual”[8].

Figure 1.3: Wireless sensors used in the daphnet project

The Nano-Tera initiative [5] at EPFL aims to improve the health and security of humans and the environment by different targets, see Figure 1.4: to detect in real time different health risks and conditions through body-integrated bio
probing, to reveal security risks through smart buildings and environments, to save energy through ambient sensing and to detect environmental hazards such as floods and avalanches from inaccessible positions on earth.

Figure 1.4: Nano-Tera initiative

Virginia Commonwealth University[9] with NASA Research partnership had developed a low-power, wireless, wearable physiological monitoring system and implemented using commercial off-the-shelf components. The non-invasive system supports physiological monitoring of skin temperature, oxygen saturation (SpO$_2$), heart rate (HR), blood volumetric pressure (BVP) and galvanic skin resistance (GSR). The sensors are integrated into a wearable device, as in Figure 1.5; it can be used to monitor health and wellness of various patient populations. Data collected by the sensors is sent wirelessly to a personal computer.

CSEM SA [1], in Neuchatel, is researching and developing to supply textiles which do more than dressing, Figure 1.6: their project will provide a warning at the raise of some pathologic states using non-invasive monitoring techniques embedded into textile, see Figure 1.7.

For example, detect to have a sufficient quantity of minerals and avoid dehydration in obese Children or sportive people, prevent the different pathologic states of patients with diabetes, which is of concern for many people and also
track the evolution of wounds to avoid the appearance of a chronic wound or an infection.

Different wearable platforms have been developed by several Universities. As the proposal in Figure 1.8 made at the University of Alabama[11]; most of them use electrocardiograms, accelerometers, blood pressure and body temperature sensor. Some are built with non intrusive sensors connected by wireless body area network to a pocket PC or similar.

Depending on the purpose, this personal computer can be connected to a home server and also do an emergency call or sending information through Internet or cell phone to the medical center.
Our project attempts to define an initial structure to perform a similar wearable health system, but able to share knowledges among user’s platform in ad-hoc mode.

Part of the process could be transferred to other near computers connected to the principal hardware; that is why we propose an agent oriented platform, taking profit of agent’s framework Jade.
Chapter 2

Project goals

The aim of this project is to define a platform to diagnose behaviours or activities. The main idea is to create a system endowed with sensors able to identify the status of the subject who wears it through the sensors signals.

The entire system should be divided in several parts, depending on their function, to simplify future improvement and addon’s development. The system is based on agents programming methodology, subsequently, each stage will contain one or more agents.

The leading stages in our system are “capture”, “pre-process”, “diagnose” and “action”. According to Figure 2.1, this platform should obtain real-time data from a set of sensors in capture stage. Then apply a pre-processing mathematical treatment to the given data in order to facilitate the diagnose task: to determine which pattern correspond to the pre-processed values. Depending on the diagnosed status, different actuations can be programmed in the action stage.

According to this and using the available sensors, an application to test the system is developed in this project. When somebody is walking, running, climbing stairs or simply sitting, his/her body emits different patterns of
signals. The demonstration consists to discriminate among repose, walking or running activities.

2.1 Capture

The capture part will manage the sensor data acquisition. It has to control the possible data applicants and distribute, in a coherent manner, the requested data to the next stage. The frequency of the acquisition data should be configurable, depending on the sensor and the need for the processing stage.

2.2 Pre-Processing

The Pre-Processing part is responsible for performing a mathematical treatment to the incoming data. The principal goal is to extract the relevant information according to the requirements of the diagnose method. This treatment can be done, for example, by statistic features like mean, vari-
ance, convolution operators and also mathematical methods like Fast Fourier Transform function.

2.3 Diagnose

Once data is manipulated, the diagnose part will decide or classify the input information discriminating from a set of patterns stored in system. The classification can be done in different methods using thresholds or signals’ correlation to calculate a measure of similarity.

The recognizable patterns can be stored in system from the beginning, but would be interesting to perform a training method to customize the patterns for different people.

2.4 Action

The Graphic user interface part will interact with the user who wears the system. It will display messages once a diagnostic is done and also receive orders or answers from the user.

Moreover, this project can be considered part of a second larger project,
where each platform will be equipped with some type of actuator in order to perform actions from diagnosis and to help one another using an exchange-diagnostic protocol between different user’s application, see Figure 2.2.

Therefore, these new functionalities should be considered different implementations in the action part.
Chapter 3

Specification

3.1 Hardware

This project has been developed and tested using an Ubidule[12] board and a pair of Toradex sensors.

3.1.1 Mainboard

The system’s core of this project is an ubiquitous computing module (Ubidule). It contains an ubichip and a X-scale PXA270 processor based in ARM architecture. An embedded Linux[10] is working over this ubidule.

It is accessible through Ethernet build-in port, Wi-fi / Bluetooth dongles and others. This board also contains five USB ports, micro SD card reader and a slot where it can be directly plugged a LCD-touchscreen.

The USB ports allow to connect sensors, human interface devices and mass storage, according on USB standards. The LCD-touchscreen in use is a Sharp
WQVGA with resolution 480x272 @ 16bpp.

The platform is modular to allow the application developer to customise his/her platform set-up. In this way the application developer can easily build his system setup by selecting what to plug into the ubidule from a set of peripherals.

### 3.1.2 Sensors

A accelerometer and Tilt sensors were available, consequently we started with them. These sensors are Oak Toradex[3] Tilt and Oak Toradex G. Both are USB 2.0 Full Speed standard compliant products working at 12Mbits/s with sufficiently performance and precision.

All Oak Sensors are built as HID devices and the report rate is adjustable from 6 milliseconds to 65 seconds. Thus driver support is built into all major operating systems.
The Oak G is a USB attached precision 3-axis acceleration sensor. Each sample contains a frame number value, acceleration X axis, acceleration Y axis and acceleration Z axis represented in 16 bits each one.

Frame number: 0 - 2048 ms

Acceleration X, Y and Z axis:

    Magnitude: 0 - 19.61 m/s² (0 - 2 g)

    Resolution: 0.0074 m/s²

Captured sensor data are transmitted through an INTERRUPT IN reports. Therefore, real time processing can be guaranteed.

- Tilt

The Oak Tilt is a USB attached precision 3-axis inclination sensor. A signal conditioning ASIC and the 3D-MEMS sensing element share the same package, thus providing the lowest possible noise and highest signal quality.
Each sample contains a frame number value, acceleration module, zenith angle and azimuth angle. Sensor data are provided in spherical coordinates, see Figure 3.3.

Frame number: 0 - 2048 ms

Acceleration: Absolute length of the acceleration vector

Magnitude: 0 - 19.61 m/s² (0 - 2 g)

Resolution: 0.0074 m/s²

Angles:

Zenith $\phi$: The angle between the positive z-axis and acceleration vector
0 - $\pi$ rad (0 - 180°)

Azimuth $\Theta$: The angle between the positive x-axis and the line from the origin to the acceleration vector projected onto the xy-plane.
0 - $2\pi$ rad (0 - 360°)

Resolution: 0.00075 rad (0.043°)
3.2 Software

Application core will be programmed in agents oriented methodology, therefore each stage in data process chain will be performed by an agent or a set of them. The communication between agents will allow to synchronize them. In other words, each agent applies the programmed proceedings to the given data and send the results to the next agent.

3.2.1 Structure

The minimum number of agents required to perform a single complete process is one for each stage. Otherwise, the pre-processed and classifying stages could be performed by more than one agent. The main structure is shown in Figure 3.4:
3.2.2 Class diagram

The conceptual model identifies the most important object classes and their association.

See file uml.pdf
Chapter 4

Activity pattern detection

During daily activities the human body emits different signals for each kind of movement. Walking, running, climbing/descending stairs or riding a bicycle are described by different signals patterns; but all of them are repetitive. We pretend to recognize these kinds of activities taking advantage of their periodicity.

For example, while walking, the knee is bent only when the foot is off the ground. At low speeds, people tend to take short steps while at high speeds, their stride is much longer.

4.1 Signal capturing

The data captured by sensors are 16-bits int values. These are originated by the movement of sensors. Accelerometer collects the XYZ acceleration modules, meanwhile Tilt collects the acceleration module and its Zenith and Azimuth angles. A frame time value is enclosed in each given sample. Figure 4.1 is a chart of the accelerometer’ values in one message while the person is walking.
Figure 4.1: Accelerometer: 256 Values X,Y & Z Axis

There are some activities which are normally longer than 1 message, that is why we wait for a second message to analyze data values. The chart in Figure 4.2 is a plot that contains a time sequence of 2 messages given from Y axis in accelerometer. Each peak is produced when the foot with the sensors returns to the ground. At these moments, the force made by the leg is added to the gravity force. The other peaks’ serie comes from the movement of the foot without sensors.
Using these steps as repetitive essential event in walking, the mean time between two steps from the sensorized leg is about 1.3 seconds and one step is done each 0.65 seconds. The function resulting from the frequency transform will show high density ranges over $1/1.3 \approx 0.77 \text{Hz}$ and $1/0.65 \approx 1.53 \text{Hz}$, see Figure 4.3.
4.2 Frequency treatment

The Discrete Fourier Transform (DFT) is an operator that converts a discrete values’ sequence $x[n]$ to the frequency domain representation:

$$X[m] = \frac{1}{N} \sum_{n=0}^{N-1} (x[n]e^{-j\frac{2\pi mn}{N}})$$

where $m = 0..N-1$

There are several methods to implement this algorithm. The Radix-2 Fast Fourier Transform (FFT) algorithm permits to compute the results in $O(N \ast \log N)$ operations. As in Figure 4.4, it takes the DFT and applies a common factor reduction equating the sum of two $N/2$ sequences to the 'N' point sequence of the original DFT. Resulting in the Radix-2 FFT Equation below:

$$X[k_1 + 2k_2] = \sum_{n=0}^{N-1} [(x[n] + (-1)^{k_1}x[\frac{N}{2} + n])W_N^{k_1n}W_{\frac{N}{2}}^{k_2n}]$$
where $W_n = e^{-j(2\pi n/N)}$

Figure 4.4: FFT Radix-2 signal flow graph

FFT algorithm requires that the number of samples in message 'N' has to be a power of 2. This number of samples and the sampling rate will determine the frequency range and the precision in the output of FFT.

According to the Nyquist theorem, “the highest frequency which can be accurately represented is less than one half of the sampling rate”:

$$f_{\text{max}} = \frac{f_{\text{Sampling}}}{2}$$

Moreover, the sampling time rate is the inverse of the sampling frequency:

$$f_{\text{Sampling}} = 1/t_{\text{Sampling}}$$
Then the frequency range is:

\[ f_{\text{max}} = \frac{1}{2 \ast t_{\text{Sampling}}} \]

The result of the FFT calculation gives 'N' complex numbers corresponding to the real and imaginary parts of the frequency component \( X[m] \). If 'N' samples are taken in the sampling interval \( T_i \), then the sampling frequency is

\[ f_s = 2 \ast \Pi \ast (N - 1)T_i \text{ (Hz).} \]

So, the frequency corresponding to index \( m \) is

\[ m \ast \frac{f_s}{N} \text{ (rad/s).} \]

### 4.3 Classification

The demonstration application tries to classify the incoming frequency data among repose, walking and running. Figures 4.5 and 4.6 represent walking and running signals while training. Both charts contain the average of data, the maximum and the minimum values collected in each frequency.

The interesting frequency range in both ‘movements’ is approximately from 0.5 to 8 Hz. The recorded information for each pattern of movement can be a simple pair of values (maximum value, maximum value’s frequency) to calculate the distance. Moreover, it also could be a vector which contains more than one signal frequency value to correlate it with the incoming data.
Figure 4.5: Walking frequency values

The classification method should operate inside this range of the FFT output values.

Once the FFT output is obtained, there are several options to perform the classification. For example, the maximum value of the incoming data could be compared with the thresholds of each recorded status. A recognized status is given when the maximum value is inside the range of the recorded data. See Figure 4.7.

It is possible to obtain a more accurate classification considering the frequency of this maximum value; to calculate the Euclidean distance between the point (maximum value, maximum value’s frequency) with the respective points of each pattern stored. See Figure 4.8.

To correlate the incoming data with the scored or learned patterns is another purpose to measure the similarity. Briefly, we have to normalize the patterns and the input vector in the frequency domain. Then, multiply the selected frequency ranges of input vector with each pattern stored. The sum of the
resulting values could determine which is the most similar pattern and classify the data.

When a classification is given; it is sent to user through the GUI and appears a message showing the status. The user can use the Training button to reconfigure the pattern of the declared status. This could be done in different ways according to the selected classification method.
Figure 4.7: Constant threshold

Figure 4.8: Linear equation threshold
Chapter 5

Design

The project has been developed in three fairly distinct parts with different methodologies:

![Methodologies Diagram](image)

Figure 5.1: Methodologies

The principal platform has been developed based in agents using Java Agent Development Framework JADE. Each agent will be loaded with a concrete functionality trying to favour modularity, according to the project’s specification.

Moreover, to introduce data values from sensors to the platform, add controllers and show results to interact with user, develop in C++ and Qt envi-
Environments have been also necessary.

## 5.1 Jade

As mentioned before, the most important role is occupied by JADE[2]. It is a tool for developing multi-agent systems fully implemented in Java source code under GNU Lesser General Public License and also complies with the FIPA specifications: employee communication language FIPA-ACL.

Jade can be considered as a middle-ware, see Figure 5.2, which implements an multi-agent platform (environment execution), a development framework (class library), and some service Agents: Life cycle, white pages, yellow pages, message transport,. . . Besides a set of graphical tools that support debugging and executing agents (RMA, sniffer,. . . )

![Figure 5.2: Jade Structure](image)

This platform can be distributed among different hosts, it is platform independent, and the configuration can be controlled through a graphical user
interface. See Figure 5.3. It can even remotely set up changes at runtime by moving agents from one machine to another when needed.

One of the most important features that JADE provide is the ability to communicate of their agents. The communication paradigm adopted, is the asynchronous message passing, Figure 5.4. Each agent has a sort of mailbox (the agent message queue) where the JADE runtime posts messages sent by other agents. Whenever a message is posted in the message queue the receiving agent is notified. Then, the agent picks up the message from the message queue to process it.

Each instance of the JADE runtime is called a container, and can accommodate many agents. The whole activity is a container platform. Figure 5.5. In each platform there should always be an active primary container which must run as the first container. The remaining containers must sign-in before beginning its implementation.

This “Main-Container” contains at least three agents, the Agent Management System (AMS), the Directory Facilitator (DF) agent and the Agent
The AMS agent represents the authority in a JADE platform. It manages the life cycle of all agents of the platform; it can create, suspend or kill agents. Also it updates a list containing the name of all agents present on the platform, providing a “white pages” service. Finally, it controls the communication channels between agents.

The DF agent records the services provided by all agents of the platform, it acts as a “yellow page” service, Figure 5.6. The ACC agent manages the communication between agents. The exchanged messages can be intra-platform and intra-container (JADE events), intra-platform and extra-container (RMI) or extra-platform (http protocol).

The agent is autonomous and is implemented as its own thread of execution. It decides for itself when reading the messages and which messages to read. Agents may require attendance and may also engage in multiple conversations simultaneously in order to run multiple concurrent tasks.

Programming an agent in Jade consist in defining a class representing our agent which inherits attributes and methods from jade.core.Agent class. Therefore, determine which behaviours it have to execute and program them. Figure 5.7.
Each agent has its own active behaviours queue. Tasks executed in this active behaviours are programmed redefining action() method.

Depending on type of behaviour, when action method ends, the agent scheduler takes out the behaviour from queue or puts it at the end again. A behaviour can be easily blocked, it is placed in blocked queue waiting for ACLMessage. If this message is received the behaviour turns again into active status and it is enqueued at the end of active behaviours queue.

Briefly, all our agents will have defined behaviours to control the agents connected in system chain, and to communicate one with each others to transmit orders and values. A manner of defining schedules priority can be configuring the time that the behaviours will be blocked in each cycle.
Figure 5.6: Yellow pages service

Figure 5.7: Agent thread path of execution
5.2 C/C++

To control the Oak Toradex USB Sensors, we started from a set of libraries working on Linux, available in the website of the manufacturer [7] which are developed in C/C++ and allow to manage and configure sensors.

Oak libraries

Source code available in manufacturer website is constituted by two c++ libraries: OakFeaturesReport and OakHidBase.

- OakHidBase: contains the device access functions, encapsulation of hidden calls into more human friendly calls such as readInterruptReport(), readFeatureReport(), sendFeatureReport().
- OakFeaturesReport: contains the implementation of generic feature report commands.

USB information is transferred in packets with a fixed structure, called “Report”. For HID devices there are two kinds of reports used:

- Interrupt report: These reports have a fixed reserved bandwidth, and therefore a guaranteed latency of no more than 1ms. All Oak Sensors use interrupt reports to transfer the measured sensor values.
- Feature report: These reports are transferred through a special control communication pipe. They have no guaranteed bandwidth thus they also have not real-time behaviour. All Oak Sensors use Feature reports to send/receive configuration parameters.

The information obtainable form sensors are:
1. Product Information

- VID (Vendor ID): this is 0x1b67 for all Toradex products
- PID (Product ID): 16 bit value which is constant for a particular product
- REV (Revision ID) 16 bit value indicating the revision status of the hardware
- SN (Serial number) This number is unique within the series of a particular product. The combination of the four parameters above leads to an identification of a USB product that is unique worldwide.
- Device Name A friendly name that describes the functionality of the USB device.
- Data Channel Information: For each data channel the USB device provides information about the data stored. Oak sensor devices always provide data in SI units. Values are reported as integers, with an associated decimal exponent, so the format is $i \times 10^{exp}$. For instance, temperatures are reported in Kelvin, distances in meters and acceleration in $m/s^2$ unit.
- Usage: What the data is used for. For common devices like keyboards, mice and game pads, the USB standard defines fixed codes for the most common functions. For sensor applications, only a few codes are defined. So the usage of in Oak sensor devices is often declared to be “vendor-defined”.
- Channel Size: Number of bits. At the time of writing this document, all Oak sensor channels are 16 bits in width.
- Value Range: Minimum and maximum valid value. This can be less than the full range that could be represented with the available number of bits. It also defines if the channel data is signed (minimum less than zero) or unsigned.
- Unit: This is a USB-standardised form to define how the channel’s physical unit is derived from the basic SI units.

- Unit Exponent: A decimal exponent that is needed to bring the reported integer sensor value into the correct range (refer to the introducing text at the beginning of this section).

- Channel Name: A friendly name that describes the content of one channel.

## 5.3 Qt

Qt[6] is a cross-platform application development framework oftenly used to develop easily graphic user interfaces known as widgets. Qt uses C++ with several non-standard extensions implemented by an additional pre-processor that generates standard C++ code before compilation.

Qt includes a rich set of widgets (“controls” in Windows terminology) that provide standard GUI functionality and introduces an innovative alternative for inter-object communication, called “signals and slots”, that replaces the old and unsafe callback technique used in many legacy frameworks.

Qt also provides a conventional event model for handling mouse clicks, key presses, and other user input. Qt’s cross-platform GUI applications can support all the user interface functionalities required by modern applications, such as menus, context menus, drag and drop, and dockable toolbars.
Chapter 6

Implementation

6.1 Capture

Two different proceedings get involved in capture chain process. The sensors are driven by c++ application and send values by socket to Jade application:

![Capture main function diagram](image)

Figure 6.1: Capture main function

6.1.1 Sensors server application

A sample application can be found hosted in Toradex web site\(^1\).

\(^1\)http://files.toradex.com/Oak/Linux/OakLinux_0081.tar.bz2
The original application is provided with a classic configure script for easy building and it reads and shows three times the sensor status.

We modified the C++ Toradex code to create a Linux sensors Server. Our application that controls the sensors, collects and packs the values to send to the Jade application by socket. It is implemented by Unix socket libraries (socket.h) opening a TCP/IP communication socket. Controlling the Input and Output message streams, it is possible to implement our own protocol to control and request information from the sensors.

![Client Server Application](image)

**Figure 6.2: Client Server Application**

Once the connection is established our C application is waiting for requesting messages by the opened socket. These messages are codified by chars:

- 'T': name (or Type) of sensors
- 'N': Number of channels
- 'M': Magnitudes [units] and unit’s exponent of channels
• 'V': channels Values of Sensors
• 'R': Modify Sample Rate of Sensor
• 'B': Modify Number of samples of buffer
• 'E': Send Exit signal
• 'S': Send Stop signal
• 'O': Ok/Ack

However, there are Java functions in CaptureConSensors.java which simplifies operation. There are three different methods of use:

1. Read information:
   (a) Type of sensor
   (b) Number of channels
   (c) Magnitudes and units of channels
   • Method:
     Client: Send char selecting operation
     Server: Get data from sensors and send by socket
     Client: Receive data and save it for future uses
     Server: Wait ack from socket
     Client: Send Ack message

2. Configure sensors:
   (a) Report rate: in milliseconds
   (b) Number of samples in message
   • Method:
Client: Send char selecting operation
Server: Send ack to client as ready
Client: Wait for Ack from server
Client: Send configuration value
Server: Check receive data and try to call requested function
Server: Send the new configured value
Client: Receive the new configured value
Client: Send Ack message

3. Read values:

This function needs an extra effort to ensure synchronization and to avoid delays, that is why we call fork() method to pack data values. Sending a message for each value read, increases communication traffic and headers, meanwhile packaging the messages introduces delay.

- Method:

  Client: Send char V to read values
  Server: Call fork() procedure
  ...
  (the part bellow will be done repetitively since ’S’ message is received)
  Server,parent: calls oak function and sends values to child by pipe
  Server,child: receives and packages data values sent splitting them between ‘;’, and sends the message to client jade by socket with N samples.
  Server: Non blocking wait for ’S’ Stop char in socket
  Client: Receive data
  ...
  Client: Send ’S’ Signal
Server: Close child
Server: Wait for ack
Client: Send Ok/Ack as operation finishes

6.1.2 CaptureConSensor.java

This class contains principal functions to operate with sensors:

- Connection: related with TCP/IP connection

  connect(String server, int port)
  disconnect()

- Read: read from sensor and save in Sensor class attributes

  readType() read type of sensors connected
  readNChan() read Number of channels of sensors
  readChanInfo() read information of each channel. (magnitude
  unit and exponent)

- Configure:

  int writeRate(int rate) modifies report rate in milliseconds
  setNsamples(int N) modifies the number of samples in message

- Read info from local variables:
There are functions to read values without connect to server application.

int getRate() returns the current report in milliseconds
int getNSamples() returns the current number of samples per message
int getTotalchannels() returns total number of channels (all sensors)

And also the public functions defined in MySensor class:

```java
class MySensor{
    String gettype() returns name (or Type) of sensor
    String getmag(int channel) returns 'Magnitudes [units]' of channels selected
    int getexp(int channel) returns unit's exponent of channel selected
    int getnumchannels() returns number of channels of these sensors
}
```

### 6.1.3 Capture Agent

The main goal of Capture Agent is to distribute the sensor’s data values inside Jade platform sending ACLMessages.

First, this agent has to register itself inside the platform informing the directory facilitator agent (DF). DF is used to search and meet other agents on the platform. This agent is registered as Type “CaptureAgent” and Name “Oak
Sensors”. Secondly, this agent has to connect to the sensors through CaptureConSensor Class and C server application. Once it is connected, it reads sensor’s information and configures default number of samples per message and their report rate. Depending of the desired activities to diagnose, another configuration can be implemented according to the necessities.

When both actions are done, this agent starts its behaviours:

- AddReceiverToList()
- CaptureRequests()
- GetOrder()
- SendValues()

All of them are extended from Cyclicbehaviour Class. That is why, it’s only necessary to program action method.

**AddReceiverToList()**

This behaviour manages the connected agents list. It’s blocked and waiting for ACLMessage configured as following:

- Performative = ACLMessage.REQUEST
- ConversationId = “AddList”
- Content Message structure “ChannAccel;ChannTitle”
  
  First param (int): Channels Selected from Sensor0
  Second param (int): Channels Selected from Sensor1

Binary expression to int: (00002 = 010 = No selected, 01002 = 410 = Channel 3 selected, 11112 = 1510 = Selected channels :1, 2, 3 & 4 )
Once a message is received, this behaviour stores in list the connected agent ID and also which channels of the sensors are selected. This information is collected in an array where each element is an instance of CaptureOutList class. There are also defined a couple of boolean flags to control the connected agent’s readiness to receive and if data saved in agent’s list is valid.

```java
class CaptureOutList {
    AID ReceiverID boolean[number of sensors] sensorsSelected
    boolean[number of channels in source] channelsSensorSelected
    boolean Ready
}
```

Then it returns an acknowledge, by creating a reply from the incoming message and configuring Performative option as ACLMessage ACCEPT_PROPOSAL and in the content’s field, the number of channels selected. In the other hand, if the message received is not understood, reply message will be sent as performative ACLMessage UNKNOWN and the content “NOT-UNDERSTOOD”

**CaptureRequests()**

This behaviour allows other agents to read sensors channel’s information and configure sensor’s values rate.

It’s blocked and waiting for ACLMessage performative CPF and the requested orders or information in the content. The incoming message has to follow the next rules:

The first char is used to select which sensor is involved in the request.
• It can be: ’A’ / ’T’ / ’B’ for accelerometer tilt or both

Next chars are used to define the orders and the information requests. The recognized requests are:

• ’T’ name (or Type) of Sensor
• ’N’ Number of channels
• ’M’+int Magnitude [units] of channel/s
• ’E’+int Unit’s exponent of channel/s

To receive related information from ALL channels, int have to be equal to 0 or equal to j e 1..N to receive the information of the j channel.

The recognized orders are:

• ’R’+int Configures rate where int is sample rate in milliseconds
• ’B’+int Configures int samples in each message

This behaviour returns a String which contains all the answers of each input request linked by ’;’ character.

Examples:

• “AT” returns name or Type of sensor “Toradex Accelerometer;”

• “BR12” configures rate and returns the new rate “12;”
• “ATM3E3R12B256” configures rate and number of samples per message and returns information from accelerator channel 3 and new rates: “Toradex Accelerometer;Acceleration y [m/s²];-3;12;256;”

• “BTNM0E0R12B256” configure rate and number of samples per message and returns information from all channels in each sensor: “Toradex Accelerometer;Toradex Inclinometer;4;4;Frame Number [s];Acceleration x [m/s²];Acceleration y [m/s²];Acceleration z [m/s²];Frame Number [s];Acceleration [m/s²];Zenith [rad]; Azimuth [rad];−3;−3;−3;−3;−3;−3;−3;−4;−4;12;256;”

Configuration commands apply to both sensors, without taking into account the first character. Even though, the inclusion of the first char used to select the sensor is mandatory, because it is read in every message.

If this agent is sending values, the configuration messages received are discarded. In this message exchange, all information fields are read from local source instead of sensors.

**GetOrder()**

This behaviour receives the orders from other agents. It’s waiting for ACLMessage: REQUEST with Conversation Id = “Order” and the order in the content.

The recognized contents are:

• “Ready” Puts the order’s sender in ‘ready to receive’ mode at agents list. Then, if capture agent is not sending, it sends start message to sensors and enables this agent to send.
• “Stop” turns the ready flag of the sender in list to false. If the message sender is the last agent connected receiving data, ‘Stop’ signal is retransmitted to sensor’s application.

• “Exit” retransmits ‘Exit’ signal to sensors, close connections and destroys itself with doDelete() method.

This behaviour, returns only a reply if Ready message is received, configured as follows:

Performative = ACLMessage.CONFIRM
ConversationId = “Order”
Content = “Done”

SendValues()

This behaviour gets values from sensor, and distributes the data. It sends a message per selected channel to each agent in list.

Sensors values are integers linked by semicolon and are received as byte’s array from socket. This array is converted to integer matrix where each column contains values from one channel. Each channel checks if any connected valid agent is requesting; therefore, if a channel is selected, the column is converted to string and then sent to the agent/s which have selected the channel.

ACLMessage’ parameters are Performative “INFORM” and Conversation Id “Values”.

50
6.2 Previous processing treatment

6.2.1 Pre-Process Agent

This Agent receives a data vector from capturer agent and applies a mathematical treatment to these received values, then sends the result to the agent which classifies or discriminate the diagnose. Capture agent is ready to establish connections with more than one pre-processing agent, then we can change targeted channel or mathematical treatment but each pre-processing agent have to be configured according to one concrete sensors configuration.

The setup() method consist in register pre-processing agent in directory facilitator as Type “PreProcAgent” and Name “FFT” referencing the mathematical treatment. Moreover it prepares local variables to run the declared behaviours:

- AddingToCaptureList()
- WaitForDecider()
• GetOrder()
• FFT()

**AddingToCaptureList()**

This behaviour searches the capture agent in the directory facilitator and try to add in its list by selecting the channel/s needed to realize the treatment. It will be executed once; according to this, the behaviour is extended from OneShotBehaviour Class. Actually, it connects to the first capture agent found; the main idea is to control all sensors by one capture agent, although this could be changed depending on necessities of future sensors.

Capture Agent identity is recorded in an attribute of PPAInList instance. It also contains information about sensor and channel selected.

```plaintext
Class PPAInList{
    AID IdAgent: ID Agent connected to this Agent
    String SensorInput: Description of data source sensor
    String ChannelInput: Description of data source channel
    int expChannelInput: Exponent of input data
}

Note: Toradex uses 10^-3 (m/s) & 10^-4 (rad)
```

This behaviour calls another method called ConfigureCapture() where it tries to reconfigure the sensor’s rates. It is not strictly necessary to apply, sensors had been configured in capture agent. If capture agent is not sending values, sensors can be reconfigured to make possible another mathematical treatment.
execution. Actually, this method is used to receive channels and sensors information and store it in PPAInList instance to inform possible connected Agents.

**WaitforDecider()**

This cyclic behaviour manages the list of agents connected, made by instances of PPAOutList class elements. It’s blocked and waiting for ACLMessage configured as follows:

- Performative = ACLMessage.REQUEST
- ConversationId = “AddList”
- Content Message = Channel or Sensor target

Once a message is received, it checks if message’s content is equal to the name of sensor or equal to sensor’s channel name. Then, this behaviour stores in the list the connected agent ID and returns an acknowledge message with “Sensor:Channel [unit]” in the content.

```java
class PPAOutList{
    AID ReceiverID
    boolean Ready
    boolean Valid
}
```

**GetOrder()**
This behaviour, as defined in capture agent, receives the orders from other agents. It’s waiting for ACLMessage: REQUEST with Conversation Id = “Order” and the order in Content.

The recognized contents are:

- “Ready” puts the order’s sender in ‘ready to receive’ mode at agents list and if it’s necessary, retransmits this order to Capture Agent and enables fft action in behaviour FFT().
- “Stop” turns the ready flag of the sender in list to false. It also checks if at least one agent is still waiting for data and, if it’s necessary, retransmits stop to capture agent.
- “Exit” retransmits Exit to capture agent and destroy itself.

This behaviour returns only a reply if Ready message is received, configured as follows: Performative = ACLMessage.CONFIRM ConversationId = “Order” Content = “Done”

**FFT()**

This behaviour is which offers the mathematical treatment. In this implementation FFT method is chosen against others to take advantage from frequency density information in values received from sensors. To use another mathematical method, a behaviour have to be developed and executed instead of this one.

As cyclic behaviour is blocked and waiting for receiving the message with values from sensors to apply the desired treatment.

This behaviour is waiting for ACLMessage defined as performative “INFORM” and Conversation Id “Values”. Then it waits for a string composed
by an integer linked by semicolon. This message is converted into a Complex Class array. The FFT algorithm returns a result in a second complex array which is sent to classifier agent.

### 6.3 Diagnose

#### 6.3.1 Classifier Agent

The main function of this agent is to classify the incoming pre-processed data, comparing with patterns stored or trained. Then, the result is sent to the connected agent, in our case the LCDAgent, to show the information to user.

The setup() method registers the agent in the directory facilitator as Type “ClassAgent” and Name “Walk/Run”. Then adds two OneShotBehaviour to search in directory and connect with pre-processing agents and the LCDAgent; besides, another two CyclicBehaviour to attend orders and to treat the received values.

- AddingToFFTList()
- AddingToLCDList()
- GetOrder()
- Classify()

**AddingToFFTList()**

This behaviour searches the pre-processing agents in the directory facilitator and tries to add in their list depending on source channel data and the
previous processing done. For example, preparing a template as Type “Pre-
ProcAgent”, Name “FFT” and content “Acceleration y \([m/s^2]\)”, this agent
will connect with the pre-processing agent who realizes FFT algorithm to
values from Y-axis in accelerometer.

When search is complete successfully, the connected agents are stored in
instance of ClassInList elements:

```java
class ClassInList{
   AID IdAgent: ID Agents connected to this Agent
   String SensorInput: Description of data source sensor
   String ChannelInput: Description of data source channel
}
```

**AddingToLCDList()**

This behaviour searches the agent LCD in the directory facilitator and con-
nects with it, sending in the content the fields’ title reported as results of the
classification in both string and value format “Title_of_String_report;Title_of_Value_report”. For example, “Status;Steps” where 'Status’ will be the title for reports like walking or running and 'Steps’ will be the number of steps done.

**GetOrder()**

This behaviour, as defined in other agents, receives the orders from user. It’s waiting for ACLMessage: REQUEST with Conversation Id = “Order” and the order in Content.

The recognized contents are:
• “Ready” retransmits this order to the connected agents connected at this agent and enables classifying process

• “Stop” retransmits stop to the agents

• “Exit” retransmits Exit to the connected agent and destroy itself.

• “Training:classification_to_learn” transmits ready order and use the incoming value to train detection patterns.

New order is added in this agent, to execute training process and to learn patterns for detection. This new recognized content is “Training:classification_to_learn” (for instance “Training:Walking” or “Training:Running”). Moreover, when this message is received, the handle of “Stop” message is reconfigured to check if agent is training or classifying. Depending on that, doLearn() method is executed to save the pattern which represents the activity or behaviour trained.

The learned data are scored on instance of class ClassData:

```java
class ClassData {
    int MeanValueAtiMin: Mean Value of minimum relative before maximum
    int MeanValueAtiMax: Mean Value of maximum
    int MeanValueVar: Estimated Variance Value
    int iMin: Index in vector of minimum relative before maximum
    int iMax: Index in vector of maximum
    String status: message to report when it is diagnosed
}
```
Training option is adding the incoming data into a vector untill the button stop is pushed. Then, the method doLearn() calculates the parameters to define a pattern of the incoming data. It averages the accumulated vector to obtain the mean frequency vector. Next, the values in class ClassData are calculated from this averaged vector.

Classify()

This behaviour receives the previous processed data and classifies it according to the patterns recorded. Therefore, it reports the results to the LDCAgent.

It’s waiting for ACLMessage INFORM with Conversation Id “Values” the content is received as Object.

As in 30 is explained, the classification could be obtained from the comparison of the euclidean distance between the point (maximum value, maximum value’s frequency) with the (MeanValueAtiMax,iMax) of each pattern stored.

6.4 Actuator

In this first approximation to the general project, the only actuator employed had been the LCD touch screen. For its use, a connection between jade and qtopia had been developed.

In this stage, an agent intervenes calling methods to interact with the qt application through an instance of LCDConQt class.
6.4.1 Gui: LCD Agent

This Agent connects the platform to qt application and interacts with user, informing about diagnostics and receiving orders from it. There are some behaviours, defined as Cyclic, launched in setup() method after registering itself in directory and connecting with qt.

Qt application methods are implemented in LCDConQt class. This class is described below.

- AddSenderToList()
- GetOrder()
- ByPassOrder()
- GetDataFromAgent()
- SendDataToLCD()

AddSenderToList()

This cyclic behaviour manages the agents connected to this agent, it’s waiting for ACLMessage performative REQUEST, with “AddList” as ConversationId and “StatusTitle;ValueTitle” as message’s content. First parameter (string) is the title for Status field and the second one (string) the title for value report.

Once a message is received, the Id of sender and the content of message is recorded in the list where each element is a LCDInList class instance.

```java
class LCDInList {
```
AID ReceiverID: Agent ID
String StatusTitle: Title of String result
String ValueTitle: Title of int result
String Status: Last diagnostic reported
String Value: Last value reported
boolean newinfo: Used to check if info had been already read
boolean Ready: Used to know if <ReceiverID> Agent its sending

Then returns a reply created from the original message configured by performative ACLMessage ACCEPT_PROPOSAL and content “Ok” as acknowledge, in other case, performative ACLMessage UNKNOWN and content “NOT-UNDERSTOOD”.

**GetOrder()**

This behaviour receives the orders from other agents. It’s waiting for ACLMessage: REQUEST with Conversation Id = “Order” and the order in Content.

The recognized contents are:

- “Ready” Puts the order’s sender in ‘ready to send’ mode at agents list.
- “Stop” turns ready flag of sender in list to false.
- “Exit” retransmits Exit signal to Qt application, close connections and destroy itself with doDelete() method.
This behaviour returns only a reply if Ready message is received, configured as follows: Performative = ACLMessage.CONFIRM, ConversationId = “Order” and Content = “Done”

**ByPassOrder()**

This behaviour checks if the socket contains not read orders and send it to classifier Agent enveloped as an ACLMessage, because classifier agent is what really attends the order.

**GetDataFromAgent()**

This behaviour receives the diagnostic message from all connected classifier agents. It’s waiting for ACLMessage: with performative INFORM and “Send Status” as ConversationId, and then “Status;Value” diagnostics in content. Then it overwrites Status and Value fields in LCDInList instance and set newinfo boolean variable to true.

**SendDataToLCD()**

This behaviour sends data from selected Classifier Agent to qt application. Each cycle checks if there is new info to show and send it to LCD socket.

### 6.4.2 LCDConQt.java

This class contains all methods needed to operate with qt application.

- Connection:
int connect(String server, int port): Connect this application to
server@port Creating sockets and read/write Streams

void disconnect(): Disconnect from Qt Application

- Sending information to show:

sendMsg(String msg) Send Message in parameter through socket
to qt application.

- Receiving orders from user input:

boolean PendingOrders(): Returns true if there are pending mes-
sages to be read.

String ReadOrder(): Read the first order in the socket buffer and
then return the message. Each order has to be ended by a
semicolon character ';' to split it from the next one.

6.4.3 Qt Application

The application developed in qtopia consist in create a widget form with the
necessaries buttons and labels, see Figure 6.4. Next, open a server socket
and waits for Jade client connection. The events from graphic elements and
socket are virtually connected to the programmed functions.

When a button is pushed, the signal of this event is used to send a message to
the Jade platform. Whereas a message is received from Jade through socket,
qt application executes another function to store the message’s content.

Start button sends ‘start’ signal and turns itself to Stop button. Training
button sends ’Training:training selected in radio button’ signal and turns
itself to Stop button. Quit button sends ‘exit’ signal to close agents in Jade and closes itself. Figure 6.5.

6.5 Demonstration Testing application

An application to test the system is provided in the sourcecode. It tries to difference when somebody is walking, running or simply sitting. Our first implementation is made by one agent in each stage. We decide to put the sensors in one leg, as in Figure 6.6, between the knee and the ankle; and then collect data from Y axis (vertical) in accelerometer.
The capture process is configured to read from sensors each 12 milliseconds, it offers a frequency sampling up to 80Hz. Values are collected individually from sensors and are sent in 256 samples per message, subsequently each message contains above 3 second of sampling data.

Our main idea is to detect some activities which are normally longer than 3 seconds, that is why we wait for a second message to analyze data values. When this second message is received, it is enqueued to the first message and fft is executed over 512 values (6 seconds of sampling data). Then a diagnostic result is done each 3 seconds but the results will contain data of the last 6 seconds.

The agent defined in Classifier stage takes the output of FFT, and searches which frequency contains the maximum relative inside the activities’ fre-
quency range. Actually, the maximum relative chosen is the second higher in all signal. In Figure 4.2 in Page 24, the mean value is above 10 m/s², that is why the absolute maximum is found in the lowest frequency (0 - 0.16 Hz), see Figure 6.7. This higher maximum derives from the Y axis signal and the constant gravity.

![Figure 6.7: Step’s frequency range](image)

Then, it compares these values with the patterns saved, calculating the Euclidean distance between the point (frequency of max value, max value) of the incoming data and learned points. Figure 6.8 shows the difference between walking and running signals. The classifier determines 'Idle' when data is under minimum, walking if the signal is greater than minimum but lesser than the running signal’s threshold.

The Gui part is programmed to show information and get orders from buttons with the LCD-touchscreen, that is why the Gui’s agent is called LCDAgent instead of GUIAgent. In our implementation, the first Classifier connected
to LCDAgent is always the selected to show information.

In improvements section, we propose one possible solution to select the function shown from each classifier agent.

6.6 Compatibility withdrawals

Sensors and Linux compatibility

Toradex sensors Tilt and G are both human interface device (HID) compliant, that is why we had to recompile ubidule kernel. The old one was compiled without HID related capabilities, concretely the new one has to accomplish these requirements: Linux kernel have to be up to 2.6 release and USB_HIDDEV and HIDRAW compatibility have to be turned on before compile it.
Process: To solve this situation uncompress linux-2.6.20-col2 sources in your Linux system. Some packages have to be installed in our system to enable compiling capabilities:

'sudo aptitude install build-essential kernel-package linux-source libncurses5-dev libqt3-dev'

Once it is installed, run 'make menuconfig’ inside linux-2.6.20-col2 folder and enable USB_HIDDEV and HIDRAW compile options, see Figure 6.9.

![Figure 6.9: Menuconfig screenshot](image)

Then save and exit the configuration menu and compile it with 'make' command. It will take some time.

After kernel is compiled is necessary to copy it in the ubidule. There is a storage flash memory partitioned in four volumes mapped in the filesystem as /dev/mtdblock0 .. 3.
The boot loader u-boot resides in first partition in flash mtdblock0, system configuration is stored in the second partition mtdblock1, Kernel image is stored in the third partition /mtdblock2 the size of this partition is above 4MB. The file system is stored in the partition mtdblock3 which 27.5MB.

Kernel image has to be copied in with:

`'dd if=<ubication/kernelfile> of=/dev/mtdblock2'`.

Once the system is restarted the new kernel is running.

User permissions:

Under Linux, devices are represented as files in the filesystem. When a device is plugged in, a special file is created in the /dev folder or in one of its subfolders (/dev/usb/ for Ubuntu). The name of the folder is of the form hiddev*, where * is a number. You may have other HID devices connected to your computer - mice, keyboards, gaming devices, etc... The application will tell you if the device you are trying to open is not a Toradex device.

By default, some distributions do not grant read/write access to the devices for standard users, only superuser are able to open them. To solve this situation, using root privileges, create a file named 10-toradex.rules in the folder /etc/udev/rules.d and write the following line in this file:

KERNEL="hiddev[0-9]*", NAME="usb/%k", MODE="0666"

`sudo echo 'KERNEL="hiddev[0-9]*", NAME="usb/%k", MODE="0666"' > /etc/udev/rules.d/10-toradex.rules`

This command will permit to normal user declared in operating system, to have access to the sensors. To solve this problem is also possible by running the application as root, or modifying access rights using chmod with
a command such as ‘chmod a+rw /dev/usb/hiddev1’. This method works but the modified access rights are lost when you unplug the device.

**Priority:**

Depending on the system it may be interesting to increase priority against the other applications in system (Jade, X11, ...). One possible solution is to add nice command to colibri to change process scheduler priority. Only root can decrease priority value (increase priority).

If an increment is given as an argument, it is used; otherwise an increment of 10 is assumed. The super-user can run utilities with priorities higher than normal by using a negative increment. The priority can be adjusted over a range of -20 (the highest) to 20 (the lowest)

**Process:**

Download busybox gnuabi source and run ‘make menuconfig ’ in the folder uncompressed. Change nice value and cross compile option: Busybox options

> Build options  ->  Cross Compiler prefix : (arm-linux-gnueabi-)

Save & exit and run ‘make’. Then copy the new executable file to /bin in ubidule and do the symbolic links to the desired applications

`root@ubidule:/bin$ ln -s busybox nice`.

**Java CVM Virtual Machine**

Virtual machine installed in ubidule is phoneme advanced mr2 working with Java 1.4.2. It’s important to keep in mind this and avoid exotic classes to develop a compatible code.

**Product:** phoneME Advanced (phoneme_advanced_mr2-b72)
Qt c-libraries

The installed Qt version is not which one we can find, actually, in sources repositories. Once compiled our qt application ask for libraries included in the folder in system path. The problem here is the path where these libraries are supposed to be. For us, a solution had been to create a symbolic link to the qtopiarm folder:

'ln -s qtopiarm qtopiarm-4.4.1'

6.7 Compilation

To compile successfully our applications, we have to follow the described instructions.

6.7.1 Java

Ubidule’s applications layer used are formed by Linux 2.6, phoneme advanced MR2 Java virtual machine, JDK 1.4.2, and Jade. To work correctly over this structure, our java applications have to be compiled in agreement to their needs:

classpath jade configured javac from JDK 1.4.2 path (not newer)
bootclass: btclasses.zip (can be included inside classpath)
Then execute in command shell:

```
javac -classpath .:/lib/jade.jar:/lib/jadeTools.jar:/lib/iiop.jar:
./lib/base64.jar:/btclasses2.zip $1 $2 $3 $4 $5 $6 $7 $8 $9
```

where $i are our java files to be compiled (./package/fileclass.java).

It compiles also all the needed classes inside our package. To avoid case sensitive problems is recommended to define the package in lower case letters.

### 6.7.2 C++

To compile successfully our sensor server application we will need the Ubideule’s tool chain installed. Then execute:

```
arm-linux-gnueabi-g++ ./oaklinuxserver2.1.cpp ./OakHidBase.cpp
./OakFeatureReports.cpp -o <outputnamefile>
```

### 6.7.3 Qt

The easy way to compile this application is getting qmake modified for ubideule, with the cross-compile option updated. Then execute from source folder in shell:

```
//path_modified_qmake/qmake -project
//path_modified_qmake/qmake make
```
Chapter 7

Conclusions

- Propose for architecture done

We had defined an agents based structure, providing each one with a set of functionalities. The system process chain is divided in four stages: data capture, mathematical treatment, diagnose and action.

The developed system’s platform is functional and is implementing all goals mentioned in the specifications. The demonstration has been tested and works fine. Framework Jade provides a clear structure to program autonomous agents and a communication protocol to send messages between agents. In our testing application, the agents are not completely autonomous, there is a dependence with the defined structure to realize the diagnose function.

Therefore, programming in agents methodology offers certain scalability and reuse. Changing few parts of source code we can do different new applications. However, to distribute our functions in several agents, instead of few agents, involves more communication traffic in the platform. This communication adds delays in every transmission between agents, that is why our application reduces efficiency. Moreover, the orders from GUI are accom-
plished after some delay. The 'start/stop' signal is retransmitted from the qt application to the sensor application through all the agents platform. These delays could be improved broadcasting some messages over the platform or reducing the time while the priority behaviours are blocked.

- Sensors Toradex values and Qt in Jade

We have provided an approach to communicate with Jade application, in order to have the values of Oak Toradex USB sensors to Jade platform and to easily interact with user taking advantage of LCD touchscreen and qtopia application. It's working

- Demonstration Application test

We developed an application to test the system platform. It takes data from an accelerometer axis and it is able to discriminate between repose, walking or running activities. Also performs steps estimation depending on the signals from the sensors.

7.1 Future Considerations and improvements

- Add more sensors:

Additional sensors could be interesting to add in our system. Oxygen in blood SpO2, an electrocardiogram, blood pressure measurer, body thermometer and also more Accelerometer/Tilt set in different points of the body. It would be interesting add wireless sensors to perform a non-intrusive system.

To add them in the system, this can be done in different parts of the code. Depending on the sensor driver necessity and the synchronization, in C application or in a second capture agent.
• Substitute diagnostic method

Actually, in our implementation calculates the distance between concrete points in signals. A possible improvement is to change it recording in ClassData instance, a mean pattern of incoming vectors and then convolute with the incoming data.

• Add second diagnostic layer

Each classifier can provide different diagnostic. This improvement consists in developing a superior agent connected between all classifier agents and Gui agents, to perform a second diagnose from each classification. For example, we could have a movement classifier, besides an electrocardiogram state classifier, then we could perform combined diagnose from both classifications.

• Exchange information system

To provide an exchange system to share information between applications of different users. This agent should be connected in the same stage than LCDAgent, use this agent when the system doesn’t identify the diagnose. The main idea is that every person wearing the system is member of an olsr network, where each member can ask / offer information.

• Agents Disconnection:

When an agent is anomalously disconnected from platform and another tries to communicate with it, the AMS agent replies with an ACLMessage informing fail on delivering the message. To add a behaviour that reads this incoming messages and turns valid flag in connected agents list to false and erase it from the directory facilitator.
• More classifiers:

In our test implementation only one classifier agent is considered; that is why in LCDAgent, the same agent is always selected. Adding tab’s widget or window in qt where user can select which function (from classifiers agents) is showing.

• Boot sequence:

Actually, is mandatory to use a concrete booting agents sequence. First to boot up are Capture and LCD Agents, which connects with external applications. Next, the pre-processing agent and finally the Classifier Agent. Wait method is programmed at the beginning of this last 2 agents to avoid problems, when all the agents in system chain are loaded at the same time. See Figure 7.1.

Figure 7.1: Agents boot sequence
This improvement proposal consists in loading first the Classifier Agent and then load pre-process agents and capture depending on the data required in Classifier/decider Agent.

- Agent Mobility:

Depending on the sensors installed, the decision methods added and the environment, when the data information traffic load is lower than its compute, some agents can be moved to another computer to improve the computational load of the system.

- Actuator Agent:

Adding another actuator agent, for example, able to call emergency services through a cellular phone or Wi-Fi connection.

- GUI Loader:

When the applications are loaded by the link in qt loader interface, there is a problem in boot and the application hangs. If we launch them with the same script by secure shell connection, the application works fine.

We talked about this problem and it seems that it was related with some referenced file in source code, because it was referenced with relative path (./dir/file). These files were used to store values from sensors during the essays. I tried to change it to absolute path (//dir1/..dirn/file) but the problem was not fixed. Then I check removing the files’ references but it was not successful. This problem is not solved.
Bibliography

[1] CSEM.  
http://www.csem.ch.


http://www.toradex.com/Products/Oak_USB_Sensors.

http://www.daphnet.eu.

http://www.nano-tera.ch.


