DEPARTMENT OF INFORMATION ENGINEERING
UNIVERSITY OF PADOVA

Università degli Studi di Padova

FACOLTÀ DI INGEGNERIA

Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Elettronica

RELAZIONE DI TIROCINIO

Interfaccia di comunicazione tra Smart Adapter e Sistema Domotico

(Gestione evoluta di un elettrodomestico via PowerModulation)

Communication Interface between Smart Adapter and Home Network

(Advanced management of a digital appliance via PowerModulation)

Student
Giovanni Costa

Supervisor
Prof. Paolo Tenti

ANNO ACCADEMICO 2012/2013

To Alice

Giovanni Costa

"Logic will bring you from A to B, imagination will bring you everywhere."

A. Einstein

Thanks

I thank my parents for letting me to go to University, even when times were not the best. Thanks to Alice, because she has had lot of patience with me on study days. Thanks to my grandparents and all my relatives, even for just a thought. Thanks to who is above us, for giving me the strength to go on every day in my study course.

Ringrazio i miei genitori per avermi permesso di andare all'università, anche quando i tempi non erano dei migliori. Grazie ad Alice per la pazienza che ha portato nei miei giorni di studio. Grazie ai miei nonni e ai parenti, anche solo per un pensiero. Grazie a colui che sta sopra di noi, per avermi dato la forza di andare avanti ogni giorno del mio percorso di studi.

Contents

1 Vimar spa **11**

2 Project Description **13**
- 2.1 Purpose . . . 13
- 2.2 Overview on the Project . . . 13
- 2.3 Grid-Aware Networked Digital Appliance . . . 14
- 2.4 Last Meter Power-Line Communication . . . 15
  - 2.4.1 Down-Stream Communication . . . 15
  - 2.4.2 Up-Stream Communication . . . 16
- 2.5 Smart Adapter . . . 16
- 2.6 Konnex Bus . . . 17
- 2.7 Smart Adapter Interface . . . 18
- 2.8 Data Type . . . 18

3 Project Results **21**
- 3.1 Overall System . . . 21
- 3.2 Hardware Structure . . . 21
- 3.3 Firmware Structure . . . 22
  - 3.3.1 Event Queue Manager . . . 23
  - 3.3.2 UART Manager . . . 24
  - 3.3.3 Smart Adapter Manager . . . 24
  - 3.3.4 By-Me Device Manager . . . 25
  - 3.3.5 Interface Manager . . . 25
  - 3.3.6 Other Reused Blocks . . . 26
  - 3.3.7 Main . . . 27
- 3.4 Interfaces with the User . . . 27
  - 3.4.1 Tablet Interface on Home Network . . . 27
- 3.5 System Debug . . . 28

4 Android Interface Application **31**
- 4.1 KNX over IP . . . 31
- 4.2 Application Structure . . . 32
  - 4.2.1 fridge_comm_interface . . . 32
  - 4.2.2 Fridge_Communicator . . . 34
  - 4.2.3 RefrigeratorInterfaceActivity . . . 36
- 4.3 Graphic Interface . . . 38
- 4.4 Application Features . . . 39

5 Design Methods **41**
5.1 Team Software Development 41
5.2 Agile Manifesto 41
5.3 KanBan 42
5.4 Timeline of the Work 43

Stage References

- **Project Title:** Communication Interface between Smart Adapter and Home Network
- **Trainer:** Giovanni Costa
- **University Tutor:** Prof. Paolo Tenti
- **Company:** Vimar spa
- **Corporate Tutor:** Ing. Alberto Pomella
- **Designer:** Ing. Dino Spiller
- **From:** 2012/09/03 **To:** 2012/11/05
- **Hours:** 180

Chapter 1

Vimar spa

Vimar was founded in the far 1945 from an idea of Walter Viaro e Francesco Gusi, at Marostica, a small town near Bassano del Grappa. It produces electrical and electronical material, e.s for domestic series, domotic systems and multiple sockets. Company name is the union of initials of **VI**aro and **MAR**ostica.

Figure 1.1: Vimar headquarters

In the 15th of March 1968, Vimar registered the patent *Sicury*, a mechanism that permits the automatic close of holes in the outlet when the plug is disconnected. It permits security - increase, so the society has given it, free of charge, to all sector producers. In the 1975 Vimar has registered another patent, *Bpresa*, able to support italian plugs for 10 and 16 amps [10].
My project has been done in the Electronical Research and Development office, which is a small part of a big company very well structured and rich in experience.

Chapter 2

Project Description

2.1 Purpose

Today, home appliances are becoming more and more *Smart*, now it's time of making them capable to communicate with the external world. Lots of houses use an internal network to improve comfort and energy saving. It was realized a device that makesd possible the communication from household appliance and the external world and is called Smart Adapter; it will be described below.
The goal of the project is to realize a communication interface between Smart Adapter and Domotic net. This realization is implemented for the Vimar's Home Network: By-Me® , and the Smart Adapter designed by Elite srl.

2.2 Overview on the Project

The Figure 2.1 shows the entire system, which contains the device developed in this project. This is a test system to demonstrate the possibility to establish a communication from one end to the other.

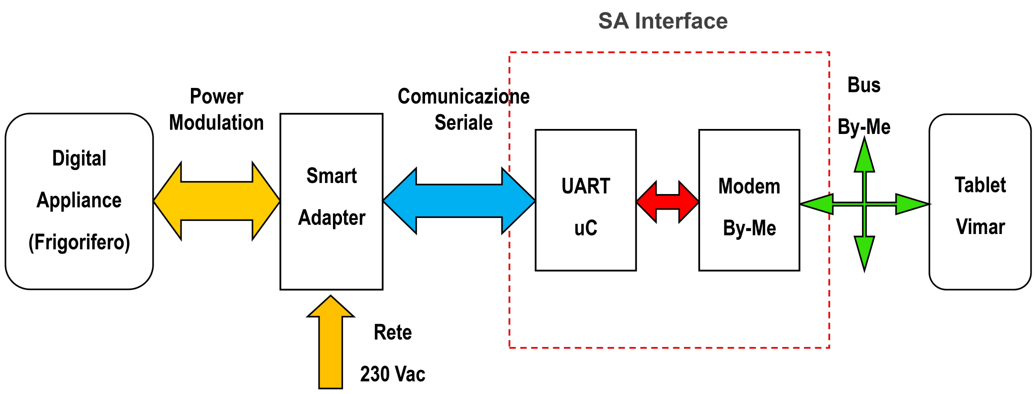

Figure 2.1: Blocks schematic of the system

As mentioned above, the Smart Adapter is a device that permits communication from and to digital appliances. Our implementation uses a simulator of a refrigerator that supports a simple Power Modulation protocol, and is made for the purpose to create a test system. The Original project uses a particular Power Modulation called *WR@P*, here isn't implemented at all, only physical layer is reproduced in full [1]. The role of Smart Adapter is to bring to the serial interface the Fridge Power Line messages and vice versa for serial ones. It's clear that this device must support the same communication protocol of the Digital Appliance. The Smart Adapter Interface is the target that project aims, here will be called *SA-Interface*. The Hardware platform uses the same of a pre-existing Vimar device. This is possible because the Smart Adapter has on board a serial RS-232, TTL standard port, also present on mentioned Vimar device, that already realized communication between serial and By-Me bus. The figure clearly shows the communication blocks of the SA-Interface. The serial side uses the internal UART of the microcontroller. The Domotic bus side uses a standard circuit, called modem, which acts as receiver and transmitter. In addition to the project finalities, it is created a user interface on a Vimar's tablet, based on Android, connected to the bus, to control the fridge or to receive messages from it.

2.3 Grid-Aware Networked Digital Appliance

In this "Smart" world also the Home Appliances need to be intelligent and able to talk with Home Networks. Lots of houses are equipped by networks that allow people to control, in a simple way, electric devices, electric loads, security systems or cooling and heating systems. The science studying Home Automation is called Domotic, merge of the latin word *Domus* and informatic. If in one side this science wants to increase the living comfort in the other it wants to reduce and improve the energy consumption in houses.
Nowadays, there is a big expansion of renewable energy sources. Home Networks and appliances must be aware that they are connected, with lots of other devices, to a distribution grid powered by several energy sources, so their development must run in parallel with the Smart Grid. It's increased the necessity to develop appliances no longer stand-alone but able to modify their functioning according to external environment consuming less energy and in a better way.
These considerations imply continuous communication with the grid or Home Network to determine if the Appliance can consume more energy or change his present state. An example of this behaviour is the Dynamic Direct Control (DDC), a continuous monitoring of frequency and voltage net. It is based on principle for which these two parameters change in correlation of the istantaneous power absorbed from the grid. Figure 2.2 shows an application example of DDC on functioning of fridge. It is shown the change of frequency and the fridge operation with or without the direct control [2].

This example demonstrated that it is possible to control operation of the Digital Appliances in correlation with the state changing of energy grid.

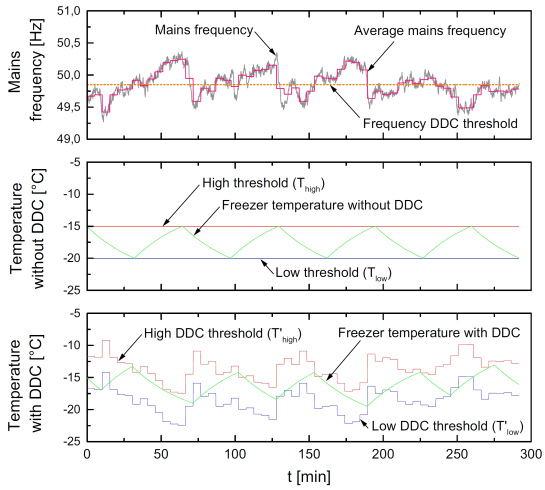

Figure 2.2: Application example of DDC Algorithm

2.4 Last Meter Power-Line Communication

The WR@P protocol realizes an Ultra Low-cost Power line (ULP) and, how it said, permits a reduction of the device costs for the last meter communication from Digital Appliance to the Smart Adapter. *Elite srl*, in collaboration with *Renesas*, has developed a microcontroller that contains the manager block for WR@P, so as to reduce hardware costs. In Figure 2.3 is shown the receiver and transmitter circuit where this microchip is used.

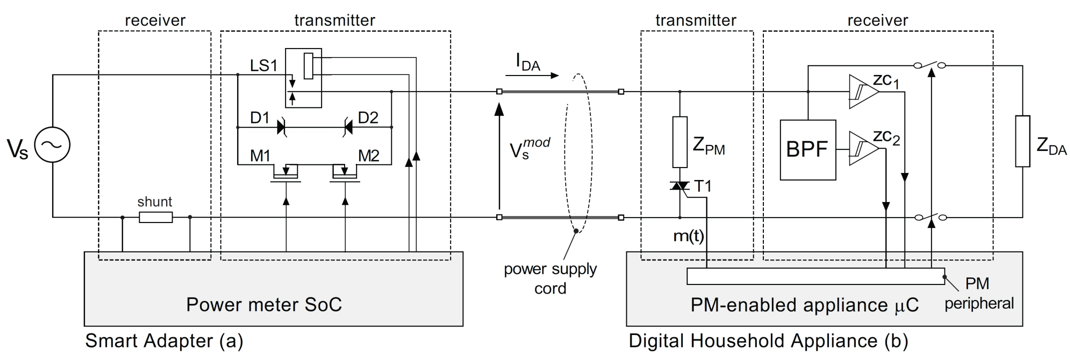

Figure 2.3: Trasmitter and Receiver for WR@P protocol

For clarity the trasmission from Smart Adapter to the fridge is named *Up-Stream*, vice versa *Down-Stream*. They have different name because of the different communication methods in the two directions [3].

2.4.1 Down-Stream Communication

The Forward link uses an On-Off Keying (OOK) modulation of the istantaneous power consumption of the appliance. The microcontroller on the fridge must be like the one mentioned before, developed for this purpose. The transmitter, into the appliance, is only made by a simple electric load, controlled throught a triac (T1), as shown in Figure 2.3.b. It is transmitted only one bit per period, this isn't a limit, because the appliance state doesn't change rapidly and requires trasmission of few bytes. The receiver only needs a power meter, each bit is decoded measuring the mean power absorbed by the appliance during each n-th cycle and compared with to an adaptive threshold, calculated as the weighted average over the k previous cycles' power consumption.

2.4.2 Up-Stream Communication

In the reverse link, where Smart Adapter is the transmitter, data bits are encoded with a Pulse Position Modulation (PPM), the transmitter generates some intentional, short and precise perturbations on zero crossing of the mains voltage waveform. Information data are transmitted four times per period, to ensure that at least one pulse is received. The multiplication of same bits is needed because the load of an alliance changes from one to another type. The perturbation isn't revealed in the first quarter of power supply period if load is inductive and if it is capacitive isn't revealed in the second quarter.
The Transmitter provides two mosfets throught which a digital device controls the current flowing into two zener diodes. A relay activates the power supply of the branch, it is shown in Figure 2.3.a. Instead, a very cheap receiver is made by a simple Analog Front-End, composed by a Band Pass Filter and a couple of Schmitt-Triggers, connected to a pair of digital counters integrated into the PM peripheral on the microcontroller.

2.5 Smart Adapter

The Smart Adapter is a general purpose communication node located at the appliance outlet. It is a bridge between the WR@P protocol and the current home networks. The development of Smart household appliances has encountered some obstacles like the absence of common home-networking standards. The appliance's hardware and firmware are indipendent from these protocols, which are managed by the Smart Adapter. In this way costs of communication, remote controlling and production testing of the appliances are not increased [4]. This solution, moreover, permits the development of new generation of Smart Plugs for the communication with the grid and others household appliances plugged. The role of the Smart Adapter Interface now is more clear, it is the missing piece for the access to a particular Home Network, in this case Vimar standard: By-Me. Note that By-Me is, however, compatible with the Konnex Standard, so Vimar's domotic bus can integrate Konnex's compatible products.
The Serial access to the Smart Adapter is based on the standard RS 232, with TTL logic levels, thus to makes simple the interfacement of an other peripheral.

2.6 Konnex Bus

Konnex (KNX) is an open, royalted and platform-indipendent building automation standard, approved by European an World certification bodies. It has been developed by *KNX Association*, from some predecessors like *BatiBUS, European Installation Bus (EIB) and European Home System (EHS)*.
The *KNX Association* is the creator and owner of *KNX Technology*, the member companies holders of rights for the certification of the interoperability of them products, that means, the capability to communicate to each other also from different manufacturer and different applications. KNX is the only standard control for home and buildings worldmate, that presents guidelines for product and training centers.
All manufacturer have to certificate compliance to ISO 9001 standard and also with the requirements of the European and International standard for Home and Buldings Electronic Systems [9]. An Example of a domotic system application is shown in Figure 2.4, where it's visible, at the bottom, a controlling central and some devices attached with it to the bus. It is possible to control lightings, plugs, home accesses and several other things.

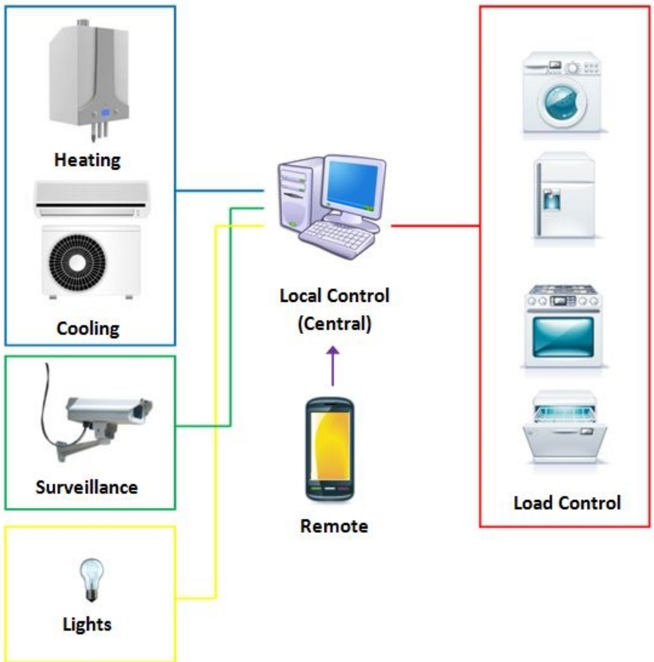

Figure 2.4: An example of a Domotic Vimar net

This KNX protocol provides some specific communication media:

- Twisted Pair (KNX TP): transmission across separate bus cable, hierarchically structured in lines and areas.
- Power Line (KNX PL): transmission on the existing mains network.
- Radio Frequency (KNX RF): transmission via radio signals, devices can be uni- or bidirectional.

- IP/Ethernet (KNX IP): can be used in conjunction with the 'KNXnet/IP' specifications, which allow the tunneling or routing of KNX frames encapsulated in IP frame.

For the project here presented a Twisted Pair media is used as principal bus, it is the most widespred in home automation. For the communication with the tablet, that is reachable only from ethernet connection, is needed the KNX IP media, the specifications are held over in '*Tablet Interface on Home Network*' section.
Note that, for simplicity, the Domotic bus here will be called only as "bus".

2.7 Smart Adapter Interface

This paragraph describes the Smart Adapter Interface, that is the aim of this project. As now is clear, it is the gateway from the serial port of the Smart Adapter and the KNX bus.
The principal device function is interpreting messages from serial and bus, after that, are required other operations to convert data and build up the frame that will be brought to the opposite side. To work in this direction it is necessary the compatibility with KNX devices, thus it should be positioned in a KNX system like all the others Vimar's KNX certified peripherals. In this specific situation not all the protocol rules are respected, because of the experimentation purpose, but the device is well integrable as mentioned above. The attention is focused on data conversion, because standards have different rules for data format and encapsulation into frames. The bigger work is in take frame payload, deduce the type and extract datas for convert them. After the conversion is constructed and sent the frame in the opposite side.

2.8 Data Type

Refrigerator makes available some information on its state and specific command to intervene on internal settings, but due to Konnex protocol rules, not all of these datas can be available in the system. Below a list of datas provided for this implementation:

- **Command:** is a code that, accordingly to the value, makes some requests or change some settings
- **Fridge Temperature Setpoint:** Value of the currente fridge temperature, it is only a sensor read
- **Status:** ia a code that represents the current state of the refrigerator
- **Trigger Setpoint:** value of the trimmer position onboard on the fridge
- **Freezer Temperature Setpoint:** value of freezer temperature as a regulation variable
- **Alarms:** code that represent a series of alarms that the fridge can comunicate
- **Energy:** value of current energy consumption

- **Power:** value of current istantaneous power consumption

Each variable is placed in a different frame and one value at time for each ones. This note is important to understand some choices in datas management.

Chapter 3

Project Results

3.1 Overall System

The system obtained from this project is shown in Figure 3.1, all parts composing the system are described in this chapter.

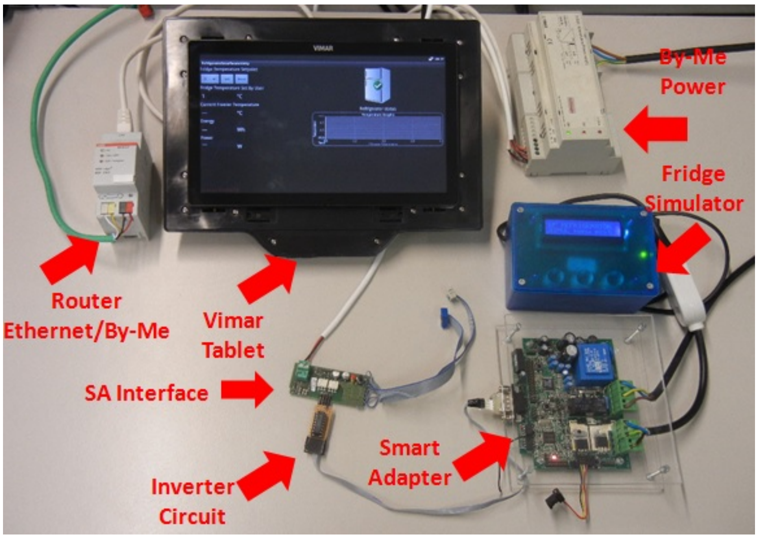

Figure 3.1: Overview on the system

3.2 Hardware Structure

The hardware circuit is the same of a Vimar's product already existing and totally compatible. Figure 3.2 describes fundamental blocks, the structure is overall simple. Follows a summary of importants hardware features.

- **Microcontroller**: ATMega 164P, 8-bit and RISC achitecture accompained by In System Programming (ISP) and JTAG Interface. Some important characteristics to underline:
  - 16KB of In-System Self-programmable Flash program memory,

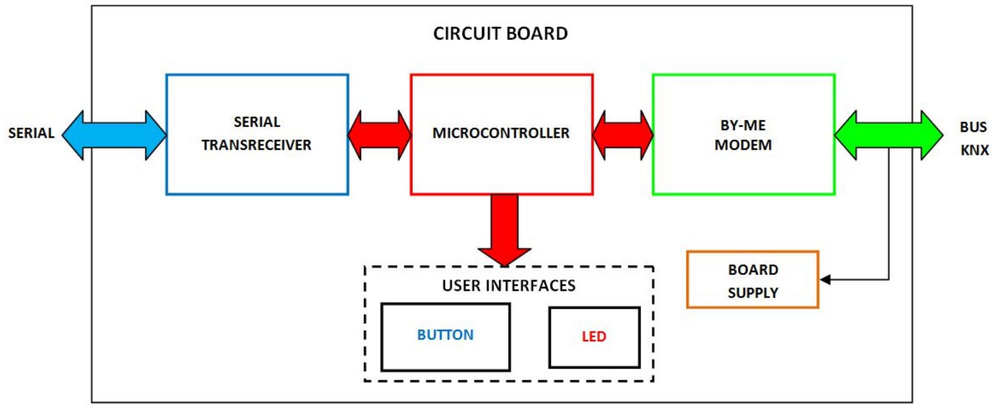

Figure 3.2: Hardware blocks

  - 512 Bytes EEPROM,
  - 1KB Internal SRAM,
  - 32×8 General Purpose Working Registers,
  - 2 programmable serial USART,
  - 32 programmable I/O lines

- **Konnex Modem**: standard communication circuit for Konnex bus.
- **Serial Transreceiver**: realizes the transmission and reception of asynchronous serial communication. Optoisolated and TTL levels compatible.
- **User Interfaces**: consists of one led and one button, necessary for the configuration in a domotic net.
- **Power Supply**: 5 Volts taken from the bus cord through a voltage regulator as LM7805.

3.3 Firmware Structure

- **Programming Language:** C
- **IDE:** *Eclipse* for C development, *IAR* to compile and *AVR Studio* to debug
- **Programmer:** *AVR Dragon*

Firmware is divided into blocks to solve single tasks or manage single peripherals, as shown in Figure 3.3. An example is the UART Manager, it has to control serial communication. This organization has lots of advantages, like: reusability of written code and not permit direct access to module data. These last can be taken calling specific functions built for the purpose in manages, in this way they are protected by overwriting. Managers can talk with other ones by throwing messages in form of events, these last are described by a specific label that represents what they mean. Firmware built in this manner, *main program* contains only: initialization of variables and registers, and an infinite loop where are continuously called all managers. Calls haven't a precise order, due to code structure. Internal structure of modules includes:

- **Process Manager:** the core of all operations, it take in charge events passed from the *main* and makes operations after
- **Initializayion:** necessary step for proper execution of operation
- **Local Functions:** according to module target

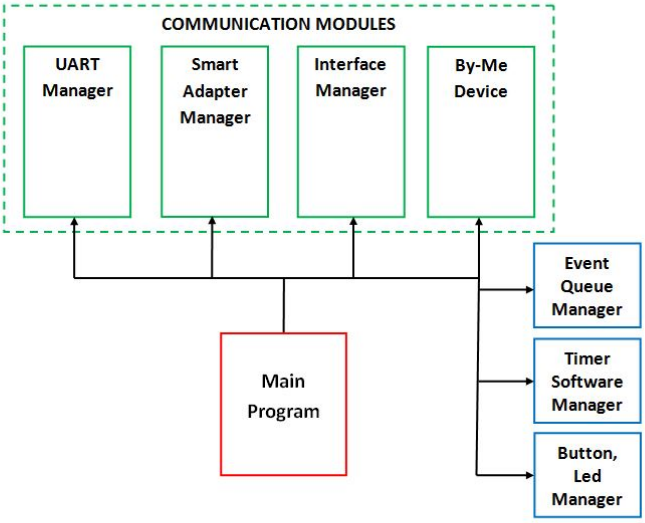

Figure 3.3: Firmware blocks

3.3.1 Event Queue Manager

This firmware block was already existent and compatible with this structure, thus is reused. All generated events are put into a circular, FIFO queue managed by the event queue manager, Figure 3.4. Also, functions for managing the queue are provided, including those devoted to: push and pop events, queue initialization.

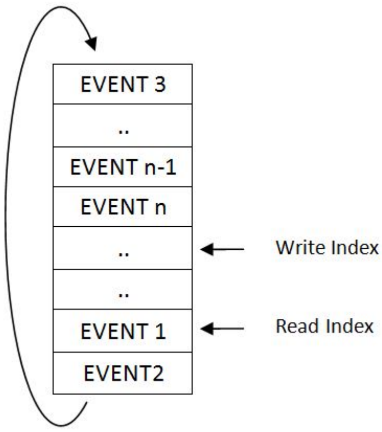

Figure 3.4: Event memorization in FIFO queue

It's important to note that since there is the possibility of a queue overflow, which can determine the loss of data or system cycles, the *Main Program* has the duty to control this dangerous situation. Events entering the queue and, one for program cycle, are extracted and passed to all managers. If they are relevant, they are processed, otherwise they are rejected. Event types are defined by an *enumeration*, which represents the numeric value of each ones. Data type *queue* doesn't exists, thus it is defined a 'C' *Structure* containing: one array and the input and output indexes, necessary for the insertion and extraction of events. An example below of these structure:

```c
typedef struct{
        int EventVect [];
        int IndexIn;
        int IndexOut;
} Event_Queue;
```

3.3.2 UART Manager

Similarly to the preceding block, UART Manager was already existing and compatible. It is responsible of: UART hardware block in microcontroller, of its configuration and its operating. The firmware module contains two buffers, one for transmission and one for reception. The first is a simple vector where is placed the frame which is going to be transmitted. Bytes are sent in polling on every program cycle. The second one is overall similar at the event queue, also here, it is a circular FIFO queue. Byte reception is in interrupt, when one is received, the UART stops program cycle, a specific function is called from interrupt routine and the byte is inserted into the queue. Initialization of firmware block is made by a particular function that permits the setting of serial parameters. Below serial configuration:

- Data bit: 8
- Stop bit: 1
- Parity: Negative
- Baud Rate: 9600

3.3.3 Smart Adapter Manager

Smart Adapter Manager has the role of: taking received data from *UART Manager* and generating the corrispondent event for the system; from the opposite side, take KNX data converted by *Interface Manager* and operate their trasmission through the UART. Principal functions are used for:

- **Initialization:** of local variables
- **Process Manager:** takes events that report message reception from UART or conclused conversion in the Interface Manager, and calls locals or globals functions to tranfere datas from the other side. For doing this datas must be extracted from the serial frame or putted into a new one.
- **Crc Check:** all received frames must be controlled over reception error through the Crc byte, that is the last one in the frame. It is calculated as the sum of all bytes of the frame on 8 bit and finally negated.

- **Build serial frames:** compone the serial frame and use UART manger's functions to put it into transmission buffer, if the transmission fail, it retries in an other program cycle.
- **Parse received serial frames:** all bytes, first the header, are decoded to generate corresponding events for the system, if one or more bytes are wrong, message is ignored and no events are generated.

Events are generated in base of frame parsing, Figure 3.5 shows the general frame structure. The header identifies frame charateristics, payload contains data and checksum is the byte for error controlling.

HEADER	PAYLOAD	CHECKSUM

Figure 3.5: General frame structure

One thing to underline in Smart Adapter functions, is the possibility to read the istantaneously absorbed current and energy consumption. They are two important parameters which allows controlling refrigerator operation. The Smart Adapter Manager generates periodically read requests for these variables.

3.3.4 By-Me Device Manager

This module contains all specific functions managing the SA Interface as By-Me bus connected device, it focus on device structure as specified by Konnex protocol. Some functions are protocol standard, but some others are specifics of the device. It is the fundamental step for bus integration, without this module, the device can't be recognized by other ones or by the Konnex Central. It isn't a real Manager like the others, simply makes available the set of fuctions described.

3.3.5 Interface Manager

It is the central node for data exchange, indipendent from any protocol, it takes events from the system and consequently, calls locals functions to process data to make them compatible with the destination protocol. In particular, when is reached the event signaling the arriving of a frame from Konnex bus, according to data type, is called function to convert this data. After that, the corresponding event is placed in the queue, like in Figure 3.6.

In a different way are finalized datas coming from SA Manager, because of the absence of a proper "manager" operating in Bus side. The *By-Me Device* module, how specified above, hasn't got an internal Process Manager, thus the *Interface Manager* has to manage the insertion of datas into Konnex frames. To do this, it calls functions in By-Me module, these, permits transmission to lower layers of the stack and so on to phisical layer, see Figure 3.7.

An important part regards Alarms management. Because of variety of these ones on the household appliance, actives alarms are memorized in the Interface

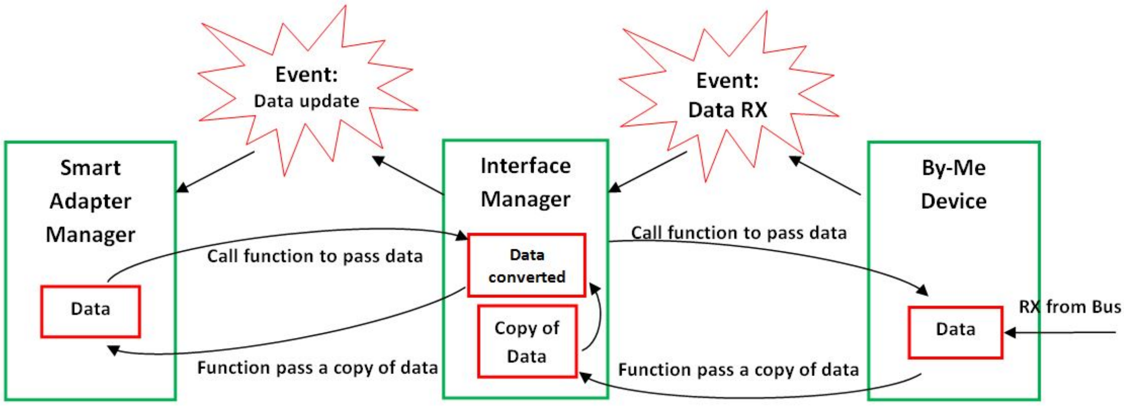

Figure 3.6: Data exchange mechanism from By-Me Device to Smart Adapter Manager

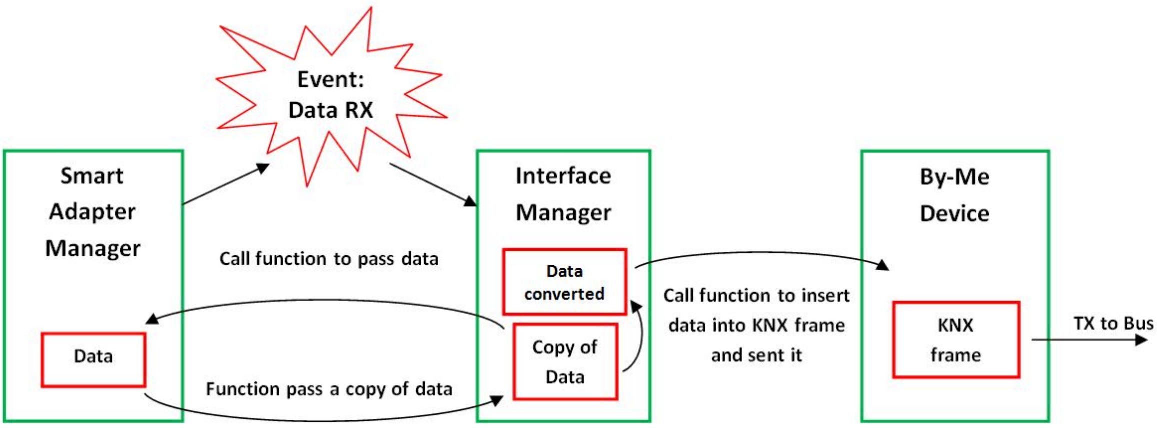

Figure 3.7: Data exchange mechanism from Smart Adapter Manager to By-Me Device

Manager. Here is decided which alarms must be retransmitted, on the basis of new alarms and resetted ones. In the bus is sent one alarm a time, so only when alarms state change, are resetted all and retransmitted updates ones.

3.3.6 Other Reused Blocks

Others reused Managers are described below:

- **Timer Software Manager:** on the basis of an hardware timer, it provides others software timer, not fine like the first but useful for not precise tasks. For example used in the auto generated event for power consumption read.
- **Buttons and Led Manager:** it controls the button and the led for device configuration in konnex systems.

3.3.7 Main

This structure Main program results very simple. It contains the initialization of variables and managers, after that, it enters in an infinite loop where there are calls to Managers. An event is extracted from *Event Queue Manager* after that it is passed to other modules. Figure 3.8 shows the mechanism just mentioned.

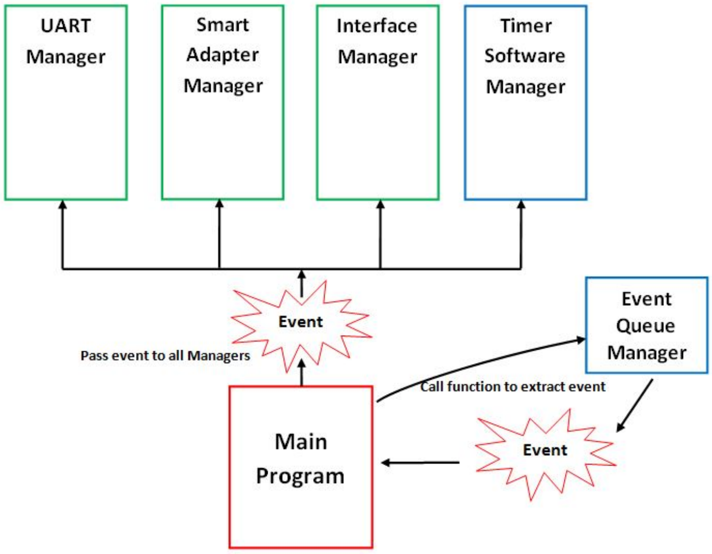

Figure 3.8: Example of event structure for relevant Managers

3.4 Interfaces with the User

The system described so far has only one point where users can interface, it is the button and led mounted on the Smart Adapter Interface. These permits the device programming like a Konnex appliance. If the interface works properly in any particular case and, in correspondance, messages sent on the bus are correct, we can say that the project purpose is reached, but how can common people understand if the system works properly? For this end, next part of the project concerns on a device to overcome this problem.

3.4.1 Tablet Interface on Home Network

By-Me domotic has already physical peripherals to visualize specific data. One of this is a tablet, chosen for this project. Below some characteristics:

- Display: horizontal 10" touch screen
- OS: Android 2.2
- Power supply 230 V a.c.
- Power supply 12 V d.c.
- LAN RJ45 port for connecting to the domestic LAN
- BUS digital line

- N/O landing push button
- SDHC port
- USB port
- Reset button

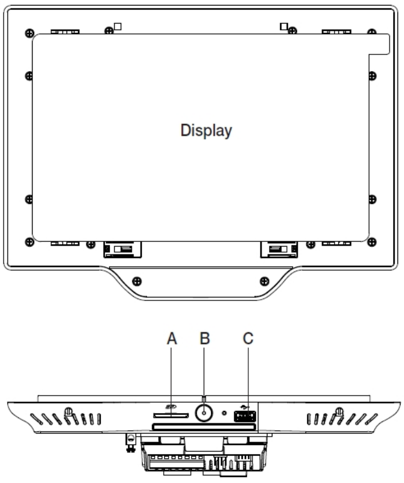

Figure 3.9: Vimar Tablet

It isn't attached directly to the bus but comunicates with an ethernet router which has the task to translate ethernet messages into Konnex ones. Tablet has to use frame syntax for Konnex over IP systems.
In this manner users can control the refrigerator from a remote position in domotic net.

3.5 System Debug

The first problem encountered has affected serial communication, because of the inversion of signals on phisical transmission in the outlet of Smart Adapter. On the SA Interface serial signal is inverted again, in this way Microcontroller's UART receives inverted levels than to expected. It is necessary an inverter circuit to adjust the link, this one is made only by two logical inverter ports, placed in both communication directions. Figure 3.9 shows this simple circuit, it is visible also in Figure 3.1 at the beginning of the chapter.

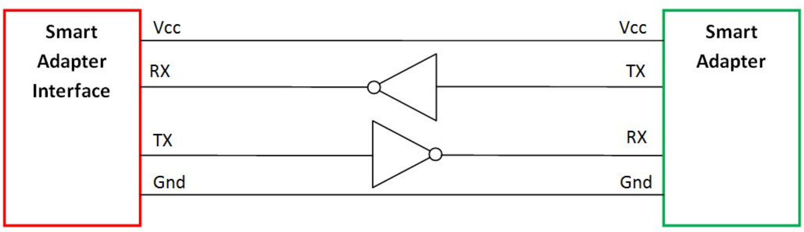

Figure 3.10: Inverter circuit for serial communication

Firmware debug is done with the support of *AVR Studio* and the *AVR Dragon* programmer using JTAG *JointTestActionGroup* port, provided in this microcontroller. JTAG permits in circuit debugging, so is simple to determine where problems are located an how resolve them. In addition, using a By-Me to USB converter and an appropriate software is possible to intercept frames on the bus and control the accuracy of messages translated by the Smart Adapter Interface. No relevant problems was encountered, only small errors was detected in the code.

Chapter 4

Android Interface Application

User can interface to refrigerator through a tablet, Vimar product. It is a flexible device which can be used for many applications. For the purpose is designed an application permitting the household appliance controls, called *Refirgerator Interface*. Below some development details:

- **IDE:** Eclipse Java, combined with Android Development Tool *ADT* plugin and Java Development Kit *JDK*
- **Software Development Kit (SDK):** 2.1 with API 7
- **Operative System:** Ubuntu 12.04, 32 bit
- **Android Programming Method:** Native Android, xml files and java classes

4.1 KNX over IP

Tablet results connected to the bus through an ethernet to Konnex converter, is understandable that it can communicate using ethernet protocol. Konnex may be used also in TCP/IP communication, for this purpose a research group of Wien University has developed an open source library called *Calimero*. It is a GNU GPL (General Public License), so it is free like any extention and program developed with it. *Calimero* provides functions to manage Konnex over IP communication, for example, it monitors link with another device (in this case the router). Below are listed the instructions to open communication with the router:

```
public int Open_Communication (String routerIP){

 int result = 0;    // Variabile per ritorno risultato apertura connessione

 try{
  netLinkIP = new KNXNetworkLinkIP(KNXNetworkLinkIP.TUNNEL,
         new InetSocketAddress(InetAddress.getByName("192.168.0.100"),0),
          new InetSocketAddress(InetAddress.getByName(routerIP),
                KNXnetIPConnection.IP_PORT),false, new TPSettings(false));
 }
```

```
 catch (KNXLinkClosedException e){
  e.printStackTrace();
  result = -1;
 }

// ...
```

It appears evident the IP addressing of router, and tunnelling option to specify protocol incapsulation. In the software are necessary only opening and closing of communication, any other operation on ethernet link is made by the library.

4.2 Application Structure

Android is based on *Declarative Programming*, programming paradigm expressing the logic of a computation without describing its control flow. Informations or data are external by principal program and are called *resources*. Those are available in ".xml" files, binary files or can be singles ones like pictures. Are all stored in a specific structure of folders, identical for any application, based on resource type, Figure 4.1.

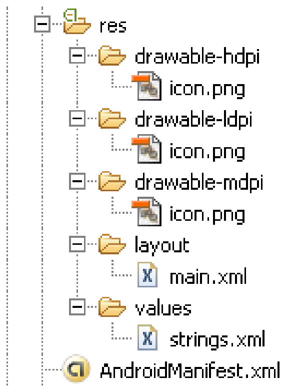

Figure 4.1: Folder structure for resources storage

Values are in different folder from icons, and those are in different folder by them resolution. Most important file is *Main.xml*, contains Graphical User Interface (GUI) structure, which components are part of it and where are placed. Application behaviour, however, is determined by a java class called *Activity*, any application must extend it, because it specify operations to be done on GUI or in consequnce to something happened on this last. It has a specific execution that reflects application ones, detailed Activity's lifecycle is described in Figure 4.2.

Application structure includes some java classes like *Activity* and necessary xml files for graphical interface, Figure 4.3 shows application blocks.

4.2.1 fridge_comm_interface

It is a java interface containing method declarations to interface with *Calimero* library. It is structured in order to implement *Callbacks*, they are defined literally

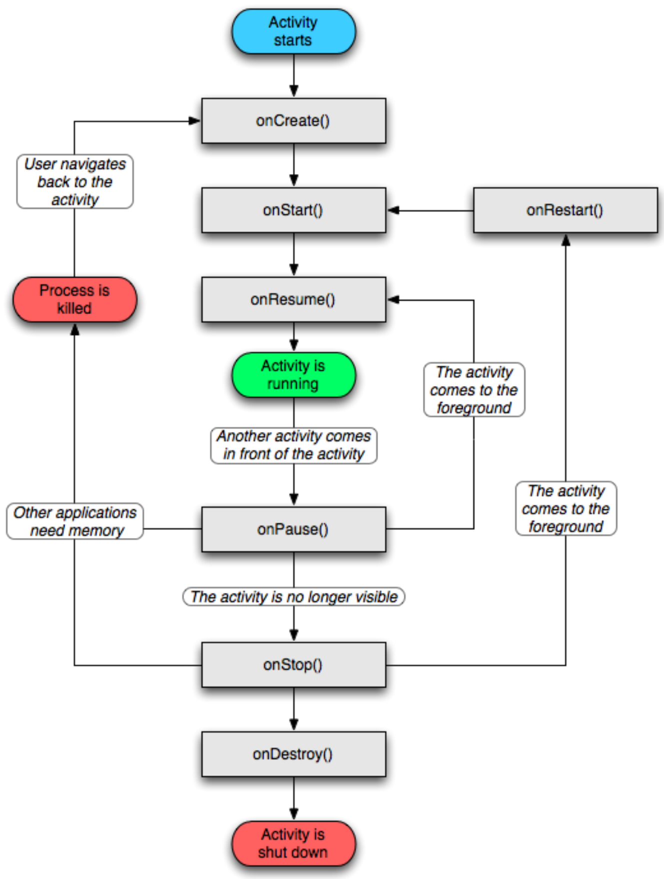

Figure 4.2: Activity lifecycle

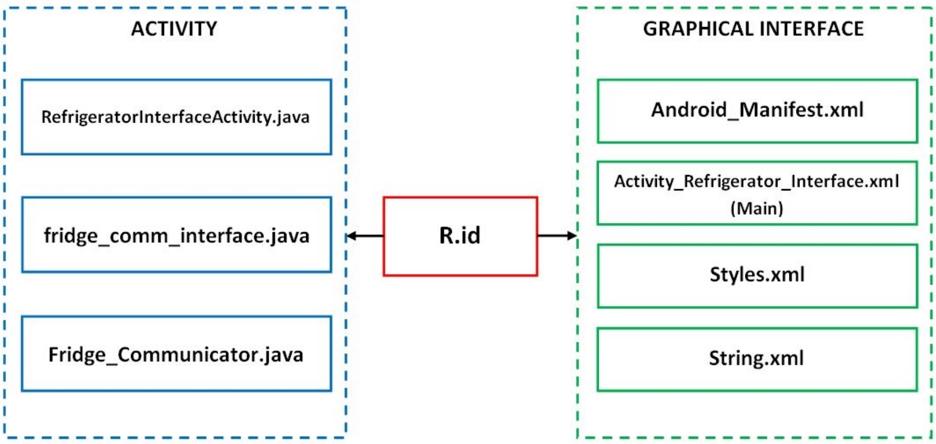

Figure 4.3: Android Application Structure

as a piece of code passed as argument to an other code. In this case, they are implemented as functions, defined in java interface but implemented in application Activity. Their task is to make some operations on Activity's scope on precise moments indicated by *Fridge_Communicator*. Figure 4.4 shows this intricated mechanism.

Below an extract from the module code.

```
public interface fridge_comm_interface {
//- ABSTRACT FUNCTIONS -
        public void on_Temp_FR_Setpoint_recv(String value_arrived);
        public void on_Status_recv(byte value_arrived);
        public void on_Trimmer_Setpoint_recv(String value_arrived);
        public void on_Temp_FZ_recv(String value_arrived);
        public void on_Alarms_recv(byte value_arrived);
        public void on_Energy_recv(String value_arrived);
        public void on_Power_recv(String value_arrived);
}
```

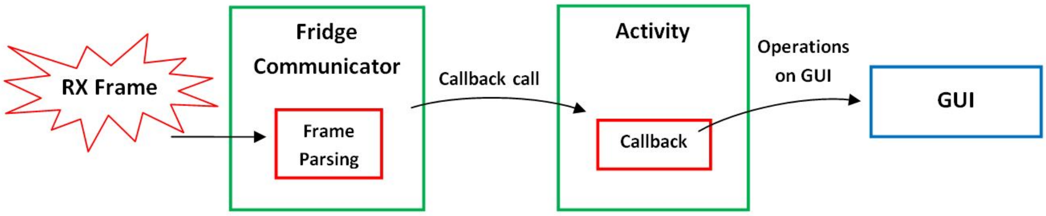

Figure 4.4: Callback functioning

4.2.2 Fridge_Communicator

It is a java class responsible of communication with the bus, it manages ethernet link with *Calimero* library methods. It implements *Thread* interface, because is necessary its backgroud running at the same time of main Android Activity. A thread is one part of a process, obtained breaking it into small pieces, executed concurrently by mono - or multi-processor systems. Here, the overall process is the application and smaller task is *Refrigerator_Interface*. Figure 4.5 describe how thread are used. In the class are defined some objects belonging to *Calimero*, used to manage communication with router. One of this is used to implement callbacks for thread, so when a message arrives from bus the callback function signals it to thread, this brings message and pass it to activity through callbacks mentioned on precedent paragraph. Below *run* method of thread, is immediate the presence of an infinite loop maintaining thread executed also when it has nothing to do.

```
public void run(){
 callb_runnable = new Runnable(){
  public void run(){
   Source_Parser();  // chiama callbacks
  }
 };

 while(true){
```

```
  try{
   sleep(100);
  }
  catch (Exception e) {
         e.printStackTrace();
  }
 }
}
```

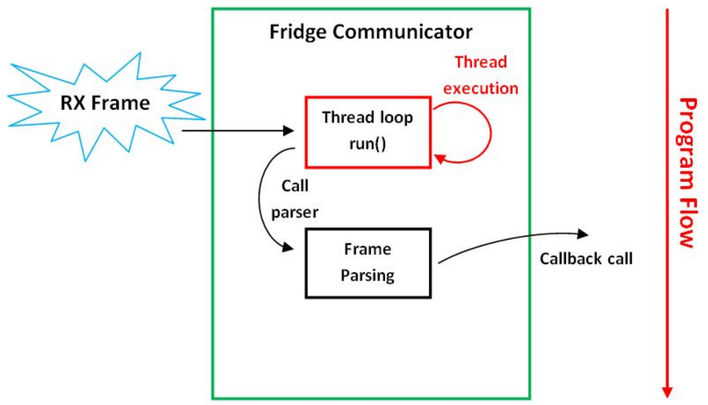

Figure 4.5: Thread functioning

Note that inside the function is defined a *runnable* object, is similar to thread but this class must be extended not implemented by an other class. This object can be called with specific functions differently from thread. More important is *Source_Parser()* method, has to parse messages and call respective callback. Following code shows class constructor, in it are visible: mentioned call instruction and listener object to capture messages from the bus.

```
public Fridge_Communicator (fridge_comm_interface arg0){

 callb_communic = arg0;
 handler = new Handler();
 listener = new NetworkLinkListener() {

  public void indication(FrameEvent arg0) {

   frame = (tuwien.auto.calimero.cemi.CEMILData) arg0.getFrame();
   frame_payload = frame.getPayload();
   source = (GroupAddress)frame.getDestination();
   subGroup = source.getSubGroup8();

   handler.post(callb_runnable); // <- RUNNABLE
  }

  public void linkClosed(CloseEvent arg0) {
  }

  public void confirmation(FrameEvent arg0) {
```

```
  }
 };
}
```

4.2.3 RefrigeratorInterfaceActivity

It is java class from which application take its behaviour, implements *fridge_comm_interface.java* and extends Activity class. Class implementation is needed from callback structure, within code are implemented methods of the interface in order to define callbacks operations towards graphical interface. Follows definition code for some callback:

```java
public void on_Trimmer_Setpoint_recv(String value_arrived) {
 textVisual.setText("Trimmer UPDATE");
 tempFR.setText(value_arrived);
}

public void on_Temp_FZ_recv(String value_arrived) {
 textVisual.setText("Temp FZ UPDATE");
 tempFZ.setText(value_arrived);
}

public void on_Temp_FR_Setpoint_recv(String value_arrived) {
 textVisual.setText("Temp FR UPDATE");
}

public void on_Status_recv(byte value_arrived) {
 textVisual.setText("Status UPDATE");
}

//...
```

When data arrives it must be communicated to the user, so firstly is displayed a *toast* (box dialog containing a text), with the instruction:

```java
textVisual.setText("Trimmer UPDATE");
```

after that are made all other operations needed by data type. For example to set new temperature received is used the instruction below:

```java
tempFR.setText(value_arrived);
```

where *value_arrived* is new data to be updated. Very interesting part concerns refrigerator doors alarm, simply described as a function that sets an animation on tablet screen. In background a thread runs to compose the animation from successives images, it is at all effects a resource for the application, in next code extract are shown mentioned instructions.

```java
public void on_Alarms_recv(byte value_arrived) {

```

```
 textVisual.setText("Alarm received: " + (int)value_arrived);

 if(value_arrived == 0){     // Ricevuto reset allarmi
  animation_freezer.stop();
  freezer_warning_active = false;
  fridge_warning_active = false;

  fridge_status_visualizer.setImageResource(R.drawable.ok);
  fridge_status_visualizer.setBackgroundResource(R.drawable.porta_ok);

  animation_freezer = (AnimationDrawable) fridge_status_visualizer.getBackground();
  animation_freezer.start();
  textVisual.setText("Reset Alarms Received");

 }
 else if(value_arrived == 3){   // Ricevuto allarme porta frigo
  fridge_warning_active = true;
  animation_freezer.stop();
  if(freezer_warning_active == true){
         //calls to resources
   fridge_status_visualizer.setImageResource(R.drawable.warning);
   fridge_status_visualizer.setBackgroundResource(R.drawable.porta_frigo_freezer);
   //thread set
   animation_freezer = (AnimationDrawable) fridge_status_visualizer.getBackground();
   animation_freezer.start();
  }
  else{
   fridge_status_visualizer.setImageResource(R.drawable.warning);
   fridge_status_visualizer.setBackgroundResource(R.drawable.porta_frigo);
   animation_freezer = (AnimationDrawable) fridge_status_visualizer.getBackground();
   animation_freezer.start();
  }
 }

//...
```

Regarding to activity life, are necessary functions describing operations to do in these situations. *OnCreate()* function point out on what to do on application start, in the opposite *OnDestroy()* for application exit. Finally the activity must catch any possible user interaction with the application, in this case it can push some buttons predisposed to send messages towards fridge. These are like call-back, an example is visible in following code:

```
public void onButtonSetTempFRSetpointClick (View view){

 int value = 0;
 String strvalue;
 int result = 0;
 String output = "Setpoint Sent";
 // aggancio oggetto spinner
 Spinner spinner3 = (Spinner)findViewById(R.id.spinner3);
 // prelevo il valore all'interno
 strvalue = spinner3.getSelectedItem().toString();

 // Converte stringa in intero
```

```
value = Integer.parseInt(strvalue);

result = fridge_comm.set_Temp_FR_Setpoint(value);

if(result == -1)
 output = "Setpoint Not Sent";
else if(result == -2)
 output = "Link Down";

// ...
```

4.3 Graphic Interface

Graphical interface is composed by these principal files:

- *AndroidManifest.xml*: contains all information about the graphical interface
- *activity_refrigerator_interface.xml*: also mentioned as *main.xml*, contains detailed description of all parts of GUI
- *String.xml*: includes definition all stings or variables
- *Styles.xml*: describes application theme

General aspect is visible in Figure 4.6, on the left there is data visualization, on the right an animation to communicate if there are fridge doors opened. Figure 4.7 shows in detail how alarms can be comunicated, for example for fridge door.

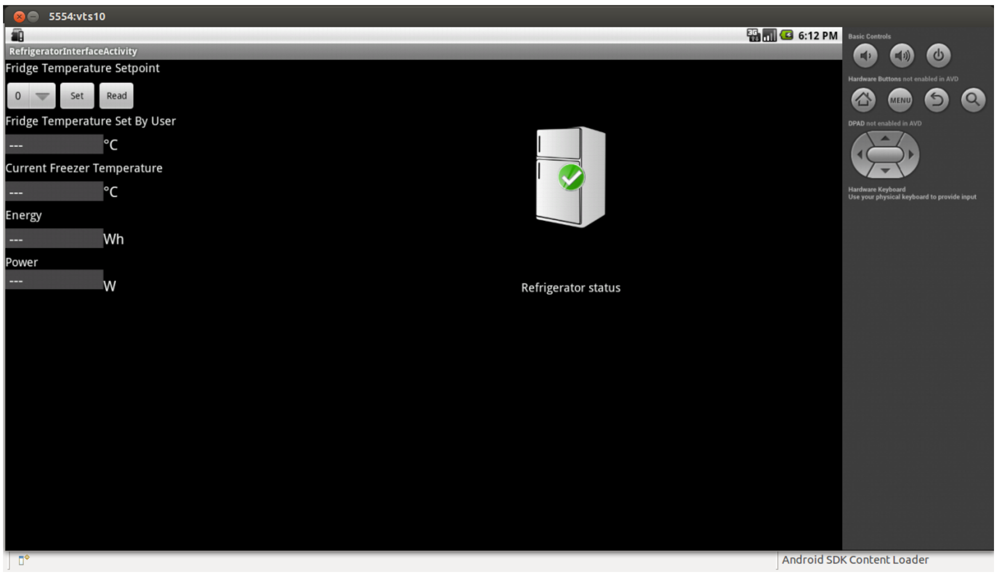

Figure 4.6: Refrigerator Interface Application

Figure 4.7: Fridge door alarm visualization

4.4 Application Features

It is interesting to underline potentials of this interface, here in a primitive state, but it makes possible many operations on household from a single remote position. Imagine a software controlling home appliances, washing machine wants to start but the oven is running and there isn't enough power available, control software doesn't give consent to start a washing cycle or it is postponed. Device results monitored in all its operations and all are comunicated to user. Next step can be in Smart Grid direction, appliance can ask it if there are conditions to absorbe power, on the other hand, grid can give a negative answer. It is immediate that this system can be useful in local control but also in bigger view.

Chapter 5

Design Methods

5.1 Team Software Development

This project has been developed with the "Agile" method, precisely, this is a group of methods for software development, that aims to minimize loss of time and uses, in the best way, know-how of team members. Agile was born in 2001 after the publication of the *Agile Manifesto* in the Snowbird, Utah resort, in a conference for discussion lightweight software development methods. It has some predecessors, from which it has been created. Some of the manifesto's authors formed the *Agile Alliance*, a nonprofit organization that promotes software development according to the manifesto's principles.

5.2 Agile Manifesto

The center of the manifesto is described below:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions *over processes and tools*
Working software *over comprehensive documentation*
Customer collaboration *over contract negotiation*
Responding to change *over following a plan*

That is, while there is value in the items on the right, we value the items on the left more.

The meanings of the manifesto's items on the left within the agile software development context, are described below:

- **Individuals and Interactions**: in agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
- **Working software**: working software will be more useful and welcome than just presenting documents to clients in meetings.

Figure 5.1: KanBan, board containing project tasks

- **Customer collaboration** requirements cannot be fully collected at the beginning of the software development cycle, therefore continuous customer or stakeholder involvement is very important.
- **Responding to change** agile development is focused on quick responses to change and continuous development.

Below, twelve principles underlie the Agile Manifesto:

1. Customer satisfaction by rapid delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Working software is the principal measure of progress
5. Sustainable development, able to maintain a constant pace
6. Close, daily co-operation between business people and developers
7. Face-to-face conversation is the best form of communication (co-location)
8. Projects are built around motivated individuals, who should be trusted
9. Continuous attention to technical excellence and good design
10. Simplicity- The art of maximizing the amount of work not done - is essential
11. Self-organizing teams
12. Regular adaptation to changing circumstances

5.3 KanBan

An useful instrument to accompain Agile is the KanBan, used in this project as fundamental tool. Big tasks are broken down into smallest ones and placed into a poster, that have the duty to create a global vision on the overall project. Figure 5.1 shows an example of the poster.

Periodically are generated these tasks that someone will take them in charge, strictly connected to the personal Kow-How. This method provide a daily meeting with team members, of about five minutes, where anyone must explain the progress of his task and if some problems have arisen. The purpose is keep update all members on the actual state of the project.

5.4 Timeline of the Work

Below are described the steps or tasks programmed during the project, from the start to the end:

1. Study of the *Elite* product (Fridge Simulator and Smart Adapter)
2. Draft of the project
3. Study of the basics of By-Me and Konnex
4. Learning the operation of hardware of previous Vimar's device
5. Drawing up documentation on project
6. Drawing up firmware Code
7. Firmware debug
8. Study of the basics of Android
9. Android application develop
10. System funtioning debug

Few words are necessary to point out the preparation of documentation on the project. Before starting any operation, were written three levels of documentation describing what will be done and how, they are described below:

- **Functional Specifications:** describes what the device can done, without technical details
- **Technical Specifications:** describes the technology used, interface to users, system architecture and eventuals insights
- **Detail Specifications:** describes, in this case, the firmware in detail

This operating method allow good conduct of the project, from the start to the end, this doesn't mean the absence of problems but when they comes you have something more to solve them.

Below the summary on the stage at Vimar:

- Number of hours provided by the University: 180
- Number of hours done: 308
- Start date: 3/9/2012
- End date: 5/11/2012
- Number of days: 40

Bibliography

[1] [A. Ricci, B. Vinerba, E. Smargiassi, V. Aisa, I. De Munari, V. Cascio, P. Ciampolini] "Wr@p: the Last Meter Technology for Energy-Aware Networked Smart Appliances", 3rd Workshop on Power Line communications, October 2009, Udine, ITALY

[2] [A. Ricci, B. Vinerba, E. Smargiassi, V. Aisa, I. De Munari, V. Cascio, P. Ciampolini] "Power-Grid Load Balancing by Using Smart Home Appliances", PM papers

[3] [A. Ricci, V. Aisa, I. De Munari, V. Cascio, P. Ciampolini] "Implementation and Test of a Power-Line based Communication System for Electrical Appliances Networking", PM papers

[4] [A. Ricci, Student Member, IEEE, V. Aisa, I. De Munari, V. Cascio, P. Ciampolini] "Electrical Appliances Networking: an Ultra-Low Cost Solution based on Power-Line Communication", PM papers

[5] [A. Ricci, B. Vinerba, E. Smargiassi, V. Aisa, I. De Munari, V. Cascio, P. Ciampolini] "Design and Test of a Microcontroller Peripheral for Grid-Aware Networked Digital Appliances", 3rd Workshop on Power Line communications, October 2009, Udine, ITALY

[6] [Brian Kernighan and Dennis Ritchie, 1988] "The C Programming Language", second edition

[7] [R.Bruni, A. Corradini, V. Gervasi, 2009] "Programmazione in JAVA", ed. Apogeo

Sitography

[8] http://www.knx.it

[9] http://www.knx.org

[10] http://www.vimar.com

[11] http://www.calimerong.org

[12] http://www.wikipedia.it

[13] http://www.agilemanifesto.org