POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Final report Smart Bracelet project

Course
IOT

Author: **Simone Asnaghi**

Academic Year: 2021-22

Contents

1 Introduction

Inside the folder containing my final project, which is my personal implementation of the SmartBracelet project, there are three folders:

- Development version, which is the folder that contains the verbose version of the project. It is one of our interests since shows correctly all debug and radio-related messages.
- Commercial version, is the non-verbose version of the project, more suitable for a commercial implementation.
- Simulations, which is the folder that contains all the simulations of the system.

Inside the first two folders there are two subfolders containing respectively the code for the first and second couple.

I chose to keep the code separated and to specify the predefined code without any automatic assignment because usually in the commercial application is a certificate to be shared, and it's easier to change the code to be ready for this behavior.

The bracelet software is developed in TinyOS and all the simulations are performed in Cooja.

2 Code

Inside every bracelet couple folder, there are several files:

- SmartBraceletC.nc
- SmartBraceletAppC.nc
- SmartBracelet.h, which is the library file that contains the message structure and the radio channel setup parameter.
- Makefile, which contains all the instructions for a correct build operation.
- Build folder, which contains all the files produced by the make instruction.

Main app files will be commented on in one of the next section.

3 Working principle

There are two possible working modes, as a parent bracelet or as a child one.

In the first case, the bracelet sends a message containing the pairing key as a broadcast message, then when receives a broadcast message from the predefined child, it stores the child's address and starts its listening phase.

Every ten seconds, in principle, it receives a message containing a mode and a position.

If for a minute it doesn't receive anything then it publishes a missing message.

The child bracelet performs the same pairing procedure as the parent one's, but when in the operative phase it generates random position coordinates and, following a probability, also a random mode.

Then it sends these pieces of information to the linked parent, till a falling message is sent.

In this case, it stops the radio.

4 SmartBraceletC.nc

This file contains the logic that controls the behavior of the bracelet.

When inserted into Cooja the parent mote will obtain an odd id whilst the child mote will obtain an even id.

My code differentiates the two cases in every sub-function of the code in order to have better debug message management.

4.1 Boot

This function is called when the mote is booted up and in both cases starts the radio interface.

4.2 AMControl.startDone

This function is called when the radio is correctly started.

In both cases, it calls a timer that resends the pairing message when one of the two motes is booted up before the other in order to have both motes ready to receive the pairing message and avoid sending this pairing message before the destination mote is correctly ready to receive that.

Also, in both cases, if the procedure returns an error the start procedure is relaunched.

4.3 AMControl.stopDone

This function is called in both cases to stop the radio when the operations are terminated.

4.4 PairingTimer.fired

This function is activated when the timer ends in both the child's mote and in the parent one's, and it's responsible to send the pairing message.

This message is sent as a broadcast one and contains the key that identifies the bracelet.

4.5 Receive.receive

This function is called by both mote types when a message is received and checks first of all if the message has the correct length.

After this first check, the message is analyzed and for each case and the single mote will perform the wanted operations.

For example, if the message is a walking type and contains the coordinates, the parent mote will display these informations on the debug message.

In the case of the child mote if the pairing message received is ok it calls the function that stops the pairing phase.

4.6 sendDone

This function is responsible for the correct acknowledgment messages interface.

4.7 WaitTimer

This function is called only by the child mote when it lost the connection with the parent bracelet and publishes a debug message, after stopping the radio.

4.8 stopPairingPhase

This function behaves differently in the case of parent mote or child mote.

In the parent case, it starts the 60 seconds timer.

This timer is used to send an alert when for a 60 seconds interval no message is received from the child.

In the child case, it sends the message to stop the pairing phase and start the operative phase.

4.9 MilliTimer

This function is called when the MilliTimer is fired.

In the parent case, this means that the child has been outside the range for more than one minute and therefore it sends a missing alert message.

In the child's case, it's used to send a message every ten seconds.

In this last case, the random function is used to generate random numbers from 0 to 1000.

5 SmartBraceletAppC.nc

This file contains the declaration of the various component used in the motes and their link with the interfaces uses in the logic code.

6 SmartBracelet.h

This file contains the library which is defined the structure of the message type used for communication procedures.

7 Makefile

This file contains the instruction for the make instruction.

8 Simulations

Inside the simulation folder, there is a .csc file that is the Cooja simulation saved.
The other three files are the log generated by simulations.

- sim.txt contains a standard simulation in which all motes are in the communication range, so the simulation ends only when a missing message is shown by both parents' motes.
- sim1_oor.txt contains a standard simulation till the first couple is no more inside the communication range. This is for testing the correct behavior of the missing message in the first couple.
- sim3_oor.txt behaves exactly as the previous one but this time is the second couple to be tested for the missing message.