POLITECNICO
MILANO 1863

Internet of Things- Final Project

Smart Bracelets

Politecnico di Milano AA 2021/2022

Computer Science and Engineering

Valeria Maria Fortina

Person Code: 10537962
Matricola: 975212

TinyOS

The objective of the project was to design, implement and test a software prototype for smart bracelets.

To do so I implemented the application using TinyOS and considered all the specifications written in the project description file. Then I used TOSSIM to simulate the program with **2 couples** of bracelets at the same time.

Here below I describe the main expect about of my TinyOS implementation.

- FILE SmartBracelet.h

In this file, we can find the structures of the messages. In particular, I used `'X'` and `'Y'` to keep track of the position of the child, the `'status'` for the kinematic, and the `'data'` for the preloaded key.

About the preloaded keys, I made the **assumption** that the file could contain a set of 20-char keys pre-loaded in the file at production time.

- FILE SmartBraceletAppc.nc

In this file, I defined all the components needed.

In particular, I created `AMSenderC`, `AMReceiverC`, and `ActiveMessageC` to use the radio.

Then I defined three Timers, one for the pairing phase, one for the child bracelet, which needs to send a message every 10 seconds, and one of 60 seconds to eventually start the missing alarm as required.

Finally the last component I used is `FakeSensorC` to provide the kinematic status of the child. And then I wired the interfaces.

- FILE FakeSensorC.nc & FaceSensorP.nc

These two files provide the kinematic for the simulation of the status of the child. In particular, I respected the requirements about the probability distribution of the kinematic status using a `rand16() % 10` function. I also used the `rand16()` function to randomly get the positions of the child.

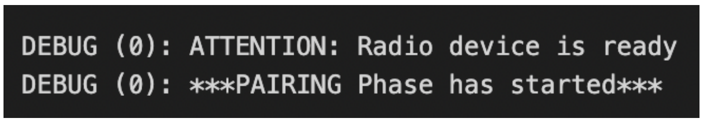
- FILE SmartBraceletC.nc

In this file I described all the interfaces and main functions to get the application working.

Here I reported a short explanation of the phases to easily understand how the code is structured. To simulate the application I used TOSSIM, using four motes. Please for the complete simulation look at the `Simulation.txt` file.

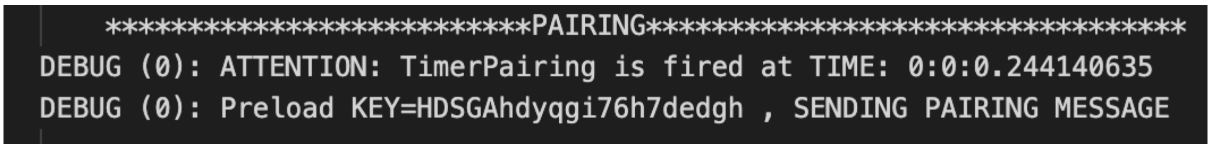
1. Pairing phase

This phase starts in `AMControl.startDone(),` each bracelet is ready to operate and the `TimerPairing` is called to start the Pairing Phase.



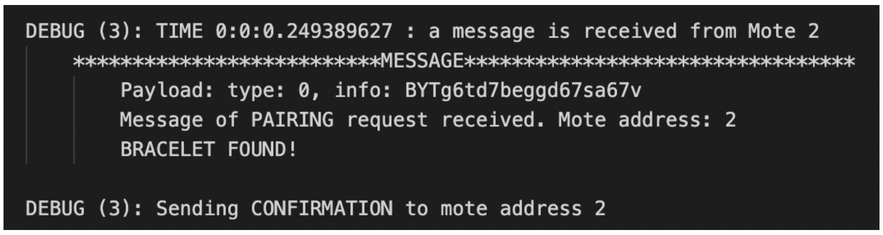
```
DEBUG (0): ATTENTION: Radio device is ready
DEBUG (0): ***PAIRING Phase has started***
```

In `TimerPairing.fired()` each bracelet sends its key to the others. To generate a couple of bracelets every two motes I used `KEY[TOS_NODE_ID/2]`. For this reason, in our simulation, we will have the first couple composed of bracelets 0 and 1, and the second couple of 2 and 3.



```
    *************************PAIRING*********************************
DEBUG (0): ATTENTION: TimerPairing is fired at TIME: 0:0:0.244140635
DEBUG (0): Preload KEY=HDSGAhdyqgi76h7dedgh , SENDING PAIRING MESSAGE
```

In `Receive.receive()` when the child bracelet receives the message from it's right pair it will send a confirmation to the other bracelet using `send_confirmation()`.



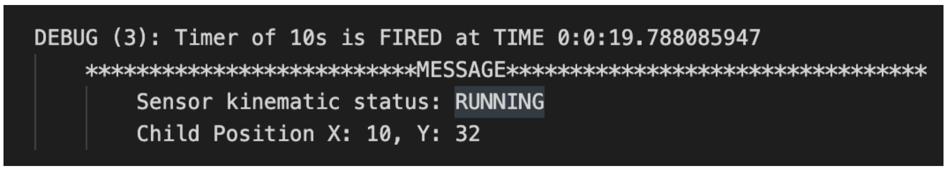
```
DEBUG (3): TIME 0:0:0.249389627 : a message is received from Mote 2
    *************************MESSAGE*******************************
        Payload: type: 0, info: BYTg6td7beggd67sa67v
        Message of PAIRING request received. Mote address: 2
        BRACELET FOUND!

DEBUG (3): Sending CONFIRMATION to mote address 2
```

As we can see in `Receive.receive()` when the confirmation message is received the `TimerPairing` is stopped and the roles of the parents and child are assigned using `TOS_NODE_ID`. This concludes the Pairing phase. And two timers start one every 10 seconds and the other of 60 seconds.

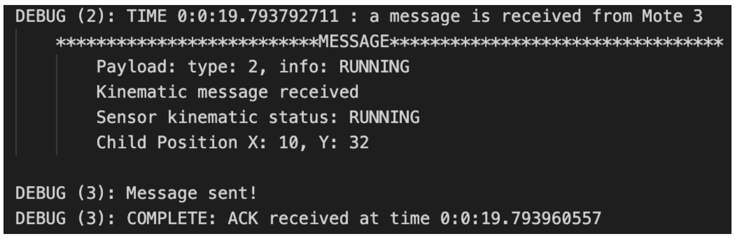
2. Operation/Exchange phase

To ensure a message exchange periodically every 10 seconds I used `Timer10s.fired()`. The child bracelet reads the fake sensor using `FakeSensor.readDone()` and the INFO message is sent using `send_info_message()`. To ensure that the only one to receive the message is the parent I used unicast with the parent's address.



```
DEBUG (3): Timer of 10s is FIRED at TIME 0:0:19.788085947
    ************************MESSAGE******************************
        Sensor kinematic status: RUNNING
        Child Position X: 10, Y: 32
```

Assumption: To ensure that the message is received I used an ACK, if the confirmation ACK is not received the message is send again.

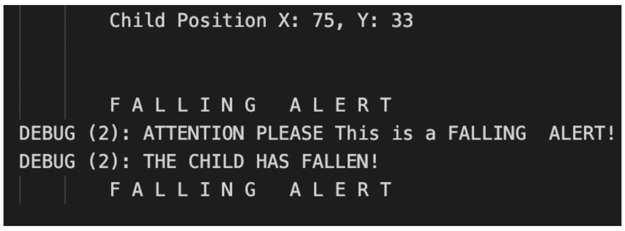


```
DEBUG (2): TIME 0:0:19.793792711 : a message is received from Mote 3
    ************************MESSAGE******************************
        Payload: type: 2, info: RUNNING
        Kinematic message received
        Sensor kinematic status: RUNNING
        Child Position X: 10, Y: 32

DEBUG (3): Message sent!
DEBUG (3): COMPLETE: ACK received at time 0:0:19.793960557
```

3. Alert mode

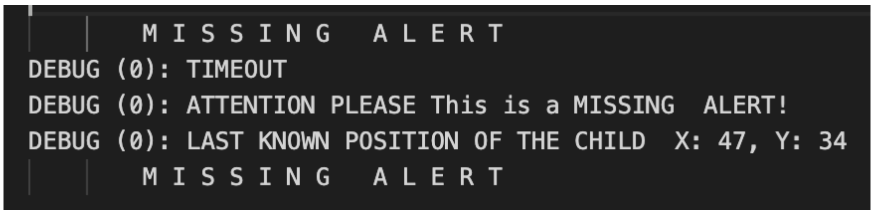
If the kinematic status is FALLING then a FALLING alert is sent to the parent bracelet as required.



```
        Child Position X: 75, Y: 33


        F A L L I N G   A L E R T
DEBUG (2): ATTENTION PLEASE This is a FALLING  ALERT!
DEBUG (2): THE CHILD HAS FALLEN!
        F A L L I N G   A L E R T
```

Finally, the second Timer simulates what happens if for 60 seconds the parent doesn't receive any new messages. I set the timer with `Timer60s.startOneShot(60000)` and I simulate the missing alert, reporting the last position of the child, with `Timer60s.fired()`. **Assumption:** After the missing alert, to continue the simulation, the pairing phase will start again and the child bracelet will broadcast its pairing key to finding the parent again.



```
        M I S S I N G   A L E R T
DEBUG (0): TIMEOUT
DEBUG (0): ATTENTION PLEASE This is a MISSING  ALERT!
DEBUG (0): LAST KNOWN POSITION OF THE CHILD  X: 47, Y: 34
        M I S S I N G   A L E R T
```