**Effort 7h**

**Component Diagram**

**Introduction**

The diagram shows the main modules that make up our system: the Java application running onto the PC and the C application running on the two Sensortags.

The communication between the PC and the Sensortag happens over Bluetooth, with the former acting as central device and the latter as peripherals. The Sensortags don’t communicate directly with each other.

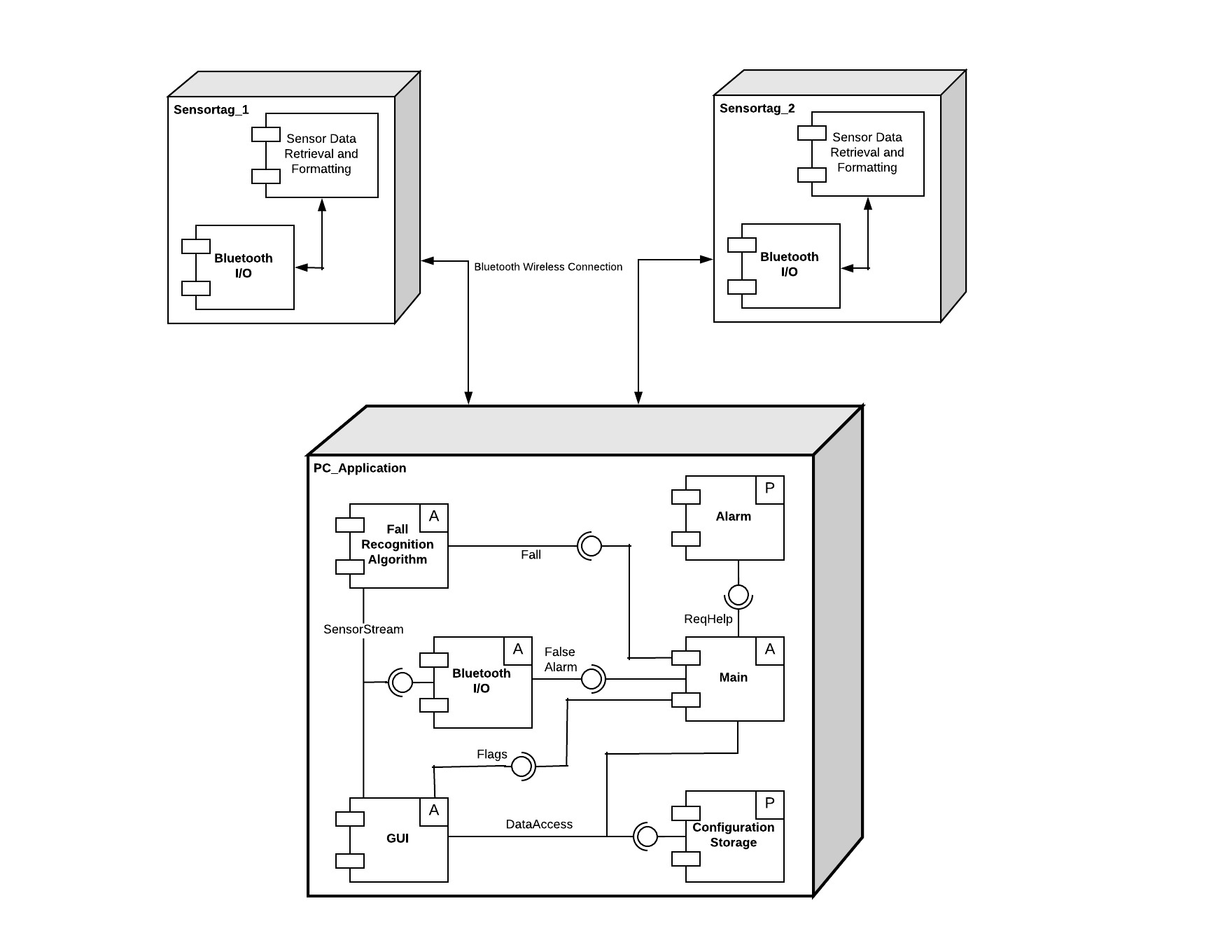


Figure 1: Component Diagram

**Sensortags**

There are two logical modules in this application:

* *Sensor Data Retrieval and Formatting*. It activates and configures the necessary sensors and actuators, receives the measured values from the sensors and formats them appropriately before passing in to the second module;
* *Bluetooth I/O.* Sets up the Bluetooth profile, services and characteristics; receives data from the previous module and forwards it to the PC\_Application.

It’s important to note that the modules described above don’t necessarily represent different files or libraries, but only different logical functions of the application, which might very well be implemented in the same file.

**PC\_Application**

The application is composed by six modules. The modules marked with an “A” are active modules, that is, they are executed within their own thread. “P” modules are instead passive, and their code is executed only when accessed by an active module.

* *Bluetooth I/O.* Manages the Bluetooth connection and receives the data from the Sensortags; sends an alarm when a Fall Event is detected;
* *Fall Recognition Algorithm.* Fetches sensor data from the Bluetooth Input module and examines it to look for Fall Events. When one is detected, it informs the GUI module and sends an alarm to the Bluetooth I/O module;
* *GUI.* Displays the graphical user interface; fetches sensor data from the Bluetooth I/O module to produce the graphs; retrieves and saves configuration data in the Configuration Storage module;
* *Main.* Contains the business logic of the application. When a Fall Event is detected by the Fall Recognition Algorithm module, raises the *Fall Detected* flag in the GUI and sends a false alarm request to the Sensortags; if the false alarm request is not answered, it reads the help data from the Configuration Storage, requests help through the Alarm module and raises *the Help Requested* flag in the GUI;
* *Alarm.* Sends an “alarm”, simulated here by sending an email;
* *File Access.* Reads and writes the configuration files.

On one hand, the separation into modules will allow the team to subdivide the development tasks and work in parallel, while on the other, the definition of formal interfaces will simplify coordination between this development efforts.

Like noted for the Sensortag application, the modules represent only a logical subdivision of the features, and don’t necessarily correspond to specific packages. However, is reasonable to expect that several of them will be implemented as independent packages.

**Interfaces**

*FalseAlarm*

Module: Bluetooth I/O

Input: control over the buzzer in the Sensortags

Output: events (presses) of button\_2 on the Sensortags

*DataAccess*

Module: Configuration Storage

Input: write access to the configuration file(s)

Output: read access to the configuration file(s)

*SensorStream*

Module: Bluetooth I/O

Input: none

Output: one data stream for each sensor

*Fall*

Module: Main

Input: none

Output: fall event detected

*ReqHelp*

Module: Alarm

Input: help message (email body); message address (email address)

Output: none

*Flags*

Module: GUI

Input: “fall detected” status (Boolean); “help requested” status (Boolean)

Output: none