Invisible Monitoring
for elderly people

CS 6235 Real-Time Systems, Fall, 2014
Report Final Project

Ramyad Hadidi
rhadidi@gatech.edu

Enrique Saurez
esaurez@gatech.edu

Georgia Institute of Technology

Contents

1 Introduction

People gets old and worried about losing their independence. Though they do not like being watched, their family members like to provide the best care for them. With today's busy world it is not possible to be near to the parents all the time. But, with the aid of technology we could remove these barriers while still let elderly people live independent. Invisible Monitoring uses Internet of Things (IoT) to connect a variety of basic sensors together without intervening the daily life of the elderly people. It is going to help their family members to watch for any suspicious event that threatens their beloved ones. Also, Invisible Monitoring will aid elderly people with basic daily chores (turning on/off lights and opening doors), reminds them of important events, and prevent an accident in its early stages.

Invisible Monitoring is a system that connects many different kinds of basic sensors to a central unit with different communication protocols, does basic chores and notify family members in emergency. All the processes are done on-line by the central unit. The purpose of these sensors could be a variety of things. The system is composed of two main part. First, is the Central Control Unit and the second part is a set of sensors which is connected to the Central Control Unit

The central unit is connected to the Internet and therefore the family member could monitor the state of each of the sensors and see the history of the activities. The central unit will notify the family member in emergency cases and messages them reports, such as a long period of inactivity in the bathroom, or too much time sleeping.The system is also going to have a record of the activities that the person usually do, and after several repetitions if there is a actuator that could do the same task, it is going to start predicting the behavior and start the action, before the person actually do it.

The sensors that are used in the system are chosen to be minimally intrusive to not threaten individuals privacy. Also, beside using the commercial sensors that are available in the market, this project tries to make and implement basic sensors. This is because first (1) commercial sensors are expensive and you need to pay a monthly maintenance cost and second (2) basic sensors like pressure or contact sensors can be easily built. Therefore, Invisible Monitoring is consisted of both type of sensors, commercial and hand-made, to prove this view that (1) you can have a smart home by combining these two sensors, and (2) for some basic information you do not need to spend a monthly cost.

The most important aspect of this project is the open source software we used to control the system. This software enables us to use many sensors with different communication protocols, control them using the tools provided by the software, and also design an user interface for controlling of the system.

In the following sections, we first restate our motivations, then in the second part we describe the system in more details beside presenting our system architecture. In the third part we describe our whole system and introduce its different parts. In this part we focus more on characteristics of each part of our

system. In the fourth part, technical details, each step of our implementation and also details of the different parts is presented. Part five concentrate on performance analysis of the whole system based on central unit performance and connectivity. Finally, in the sixth section we give a short description of other endeavors in this area, including academic and commercial.

2 Motivation

The project has two main motivations. The first one is helping elderly people. The second one is contributing to AwareHome Research Initiative at Georgia Tech.

2.1 Aiding Elderly People

> *"To all, I would say how mistaken they are when they think that they stop falling in love when they grow old, without knowing that they grow old when they stop falling in love."*
>
> *Gabriel Garca Mrquez*

Eventually, everyone gets old. We heard so much about horror of aging, from sudden aches to loneliness. But truth to be told, old ages is time to get satisfaction from your life. Seeing your kids succeed in their life and be cherished in their hearts. Find free times to finally do your favorite things, travel, reading, painting, and ....

Living at home – specialists call it aging in place – is what most people want for their later years. Americans 40 and older are just as worried about losing their independence later in life as they are about losing their memory. Common-sense interventions like grab bars in bathrooms and taping down rugs to prevent tripping can make homes safer as seniors deal with chronic illnesses. However, there is a rare chance of unforeseen happenings, and everybody wants to be near to their beloved ones to help them while these events occur.

Technology makes its first steps in designing monitoring systems for elderly people. Already, some companies are offering monitoring packages that place motion sensors on the front door, a favorite chair, even the refrigerator, and then send an alert to a family member if there's too little activity over a certain period of time. Other gadgets can make pill bottles buzz when it's time for a dose and text a caregiver if it's not taken, or promise to switch off a stove burner that's left on too long. But, these companies are so scarce and their services cost like 500$ per month.

On the other hand, elderly people do not like being watched and loose their independence. But, family members like to monitor their beloved ones to know their conditions or see early signs of a disease and simply do something. Invisible monitoring for elderly people tires to achieves this goal and keep both side

satisfied. Give elderly people their independence while giving every piece of valuable information to their family members in order to improve their life conditions. Invisible Monitoring also helps elderly people do their daily basic chores and prevents a simple negligence become an accident.

2.2 Contribution to AwareHome

Begun in 1998, the AwareHome Research Initiative at Georgia Institute of Technology is an interdisciplinary research endeavor involving faculty and student researchers from multiple domains across campus. The AwareHome building was designed with two identical floors, each featuring: a kitchen, dining room, living room, 3 bedrooms, 2 full bathrooms, and a laundry room. But it was also designed with many features not found in a typical home in order to support the expected research to be conducted.

Our project will provide a testbed for future projects in AwareHome. Currently, AwareHome connects different sensors and actuators with different commercial central devices. These devices are expensive and require a monthly cost for maintaining. Also, they are closed source so we cannot connect custom sensors to them. In addition, commercial sensors are designed to connect to a particular central unit. This will result in a problem of multiple independent sources, that are more complicated to integrate into a fully automate the home.

Our purpose is to implement a central unit on a computation platform. We will use an open source software and modify it in order to plug different sensors, custom or commercial, and support different data packages structures. This project will provide a testbed for AwareHome to implement more custom sensors and also extend their automation more easily in their research.

3 Description

Invisible Monitoring is a system consists of many parts. Therefore we present system architecture diagram and a short description of each part. The system architecture consists of seven main parts.

Sensors Sensors is the basic components of the system. They measure and sense their environment and send data to the central unit. Also, it is possible that some sensors are connected together with a specific protocol (e.g. Zeegbee) and the head node receives all the data from the other nodes and transfer all the data to the central unit.

Actuators Simple actuators are implemented in the system to show how an event can result in an act. Actuators can be connected to the central unit with different protocols. Their activation can be defined with combination of different events or data.

Software The core of the system which will collect the data from the sensors. Then stores the data received on the database and starts a corresponding action either in the actuators or alert system.

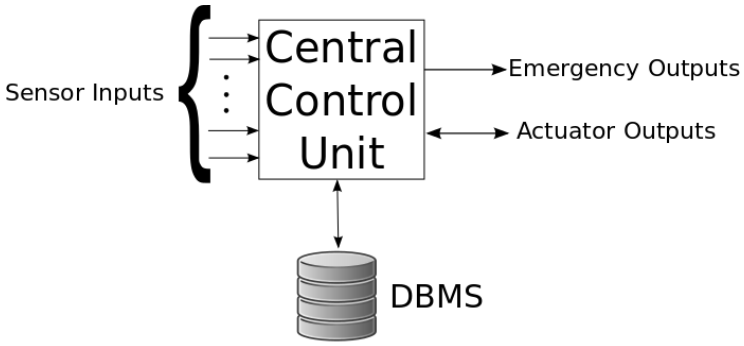

Figure 1: System Architecture Diagram

Alert System This system is implemented in the main unit. It task is to determine if something is happening an alert the individuals about it. This alert might be a SMS, a call, or a sound alert for the person. The main algorithm is as below. [21]

1. on <event expression>
2. if <condition>
3. do <action>

Database The database is used for data storage and also for accessing these data from the Internet.

User Interface A user interface is designed to access the database's data from the Internet. The implementation is a web page.

Binding The bindings are the abstraction that integrates an external system like a service, a protocol or a single device, similar to a driver but in a high level of abstraction.

Diagram 1 is included which shows our main parts of system architecture. A closer view for the central control system is shown on figure 2.

In following subsections we investigate our central unit and sensors in more details.

3.1 Central Control Unit

We implement our central unit on BeagleBone Black[2]. The selection of BegaleBone Black over the other computation platforms such as Raspberry Pi[10] was because the resources we have. In reality there is no difference between using any specific platform. The only constraint is that the proposed platform need to be able to run java[6] machine on it because our software for controlling the system is written in java. The software we choose is OpenHAB[9] and it is an open-source software. In following subsections we will introduce the main two part of our central control unit.

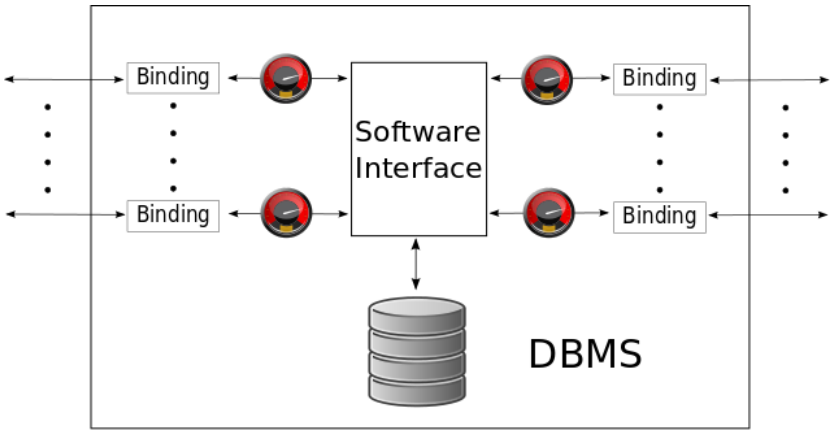

Figure 2: Central Control System

3.1.1 BeagleBoneBlack

"BeagleBoneBlack is a low-cost, community-supported development platform for developers and hobbyists".[2] BeagleBonBlack has a AM335x 1GHz ARM Cortex-A8[1] processor, 512MB of DDR3 RAM and a 4GB 8-bit eMMC on-board flash storage. It also has ports like USB, Ethernet, HDMI and GPIO which is organized into 2x 46 pin headers. Operation systems that could be run on the BeagleBoneBlack is Debian, Android, Ubuntu, and many more other. In this project we used the Debian operating system.

3.1.2 OpenHAB

The open Home Automation Bus (openHAB) is a free and open source software project. It is an integration platform for the smart Homes, which means that it makes it possible to connect to different kinds of classical home automation systems, such as KNX, Z-Wave, Insteon, EnOcean and others. Additionally, it integrates all kinds of new Internet of Things (IoT) gadgets and devices. Today over 80 different technologies are supported. Thus, openHAB allows overarching use cases by providing automation rules and uniform user interfaces across system boundaries. Data flowing through the platform can be persisted to both internal (rrd4j,...) and external (sen.se) data warehouses, which permit to create intelligent applications and integration with third party platforms (e.g. Internet of Things).[4]

openHAB is a pure Java solution and uses OSGi for modularization. Its focus is to be vendor-neutral as well as protocol and hardware agnostic. A design goal is to be able to work off line, i.e. that no cloud service or Internet connection is required to use the system, but that it forms an "Intranet of Things"[3]. openHAB has won the IoT Challenge 2013[1].

[1] `http://www.ti.com/product/am3358`

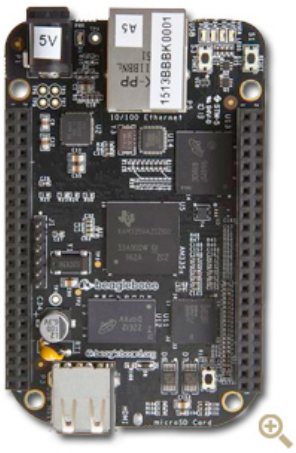

Figure 3: BeagleBoneBlack

Figure 4: openHAB logo

openHAB can be split in two parts:

1. openhab-runtime: Which is the core functionality of the program. It is the server which does all of the work.

2. openhab-designer: This is and IDE for users to check the syntax of their code and help them to configure openhab-runtime.

The main part of the opehHAB is runtime, and therefore we focus of its details.

3.1.2.1 openhab-runtime

The openHAB runtime is a set of OSGi bundles deployed on an OSGi framework (Equinox). It is therefore a pure Java solution and needs a JVM to run. Being based on OSGi, it provides a highly modular architecture, which even allows adding and removing functionality during runtime without stopping the service.

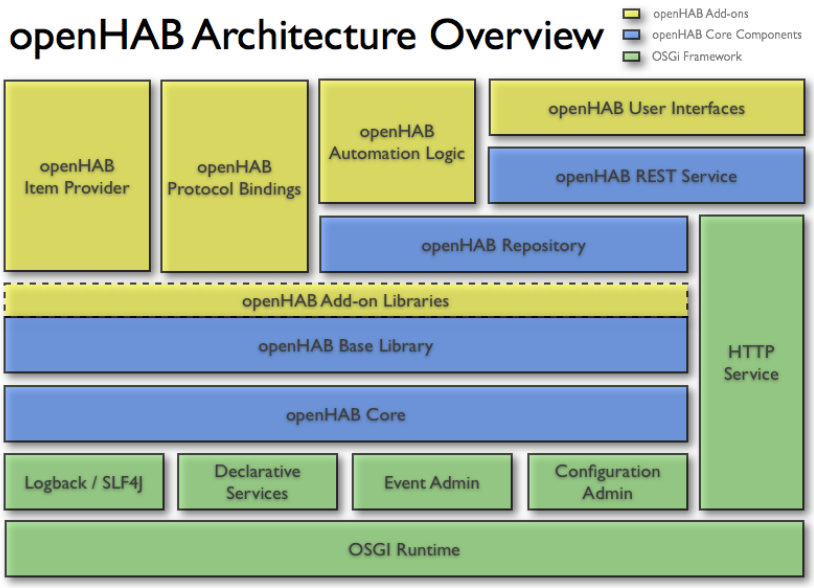

Figure 5: openHAB Architecture

Figure 5 shows an overview of the main bundles and how they depend on each other.

openHAB runtime uses a set of files for configuration. These file can be updated, added or removed when the core is running. The code will identify these changes and updates its values. These files are stored in the `configurations` directory under their corresponding directory. Here is the list of the types of these files.

items An item is the main structure that abstracts the functionality of the sensors, it makes all different sensors protocols behave equally. This includes Internet protocols, serial port, etc.

rules using the data provided from the items abstraction we are able to define rules for initiating external functions, send messages, activating actuators or any action related to the protocols supported by openHAB.

persistence This is the main file to describe the process of storing information from any state inside openHAB (item) into a *persistant* database. It has many options, but the most important are MySQL and rrd4j.

sitemaps This is the file that contains the structure of the webpage, using the group of items, and any state described as an item, we can show and control the state of all the items defined in the code.

transforms This is used for translating the information coming from the sensors into more legible words that humans understands. Like translating

zeors to closed for door sensros.

3.1.2.2 HABmin

HABmin is a web administration console for openHAB. It provides a complete interface for administering openHAB, including the following features:

- General configuration
- Configure bindings
- Configure items
- Configure mapping
- Configure sitemaps
- Configure ZWave network
- Configure rules and notifications
- Query and graph data from persistence stores

All of these properties is an add-on to the openHAB. In section 4.1.5 we will provide how to execute HABmin with openHAB. Also, for configuring various Z-wave devices we have used HABmin (see section 4.3)

3.2 Invisible Monitoring Sensors

In this phase of the project different sensors will be studied based on their protocol and how intrusive they are. Then, we will connect them to the central unit. Also, we will define rules based scenarios for controlling these sensors. These scenarios will use real-time data from sensors to control actuators.

Below is a list of different sensors and their main purpose of each one.

Pressure Sensors These sensor could be implemented on floor, bed, or favorite chair. With these sensors the presence of the the person could be monitored besides the time he/she spends there. If a certain time of inactivity detected, the system could give a message to an emergency contact person. Also, the history of presence is available which could be helpful in detecting early signs of a disease.

Lights Modules With a controllable lights it is easy to turn on or turn off the lights and adjusting its intensity.

Door Sensors Simple sensors could monitor doors and detect the presence of the person in combinations with pressure sensors. Additionally mechanical motors could be implemented to help the person in opening, closing or locking the doors at night or when nobody is at home.

3.3 How to Detect Individuals?

For every room we have a set of sensors. These sensors trigger some intermediate variables when they detect any changes. In this way, we can integrate all sensors in a common activity detection variable. Currently, our system detects all kinds of motions. Therefore, a possible future work could be detecting only humans and not other movements such as pets.

4 Technical Details

In this section we present detailed technical procedures as a reference for future works. These details includes how BeagleBone Black can be used as the central unit, how to configure various Zwave and ZigBee devices, and how to configure openHAB with all of these units.

4.1 BeagleBoneBlack

As we describe in subsection 3.1.1, different Unix-based operation systems can be run on BeagleBone Black. We used Debian distribution of Linux for our work because first, it is supported by BeagleBone Black and second, it has a good community and online resources. In the following subsections we are gonna describe each step we have done in order to make BeagleBone Black our central unit.

4.1.1 Installing Linux

Since the on-board storage of BeagleBone Black is limited to 4GB and also almost every distribution of Linux will fill up this space, in order to have more free space and also have a unified file system, we implement all of our system on a microSD card with the size of 8GB. By using a microSD card, we could easily make a backup of our system and also change the files directly with computer. Therefore we never used on-board e-MMC in this project, but it is available as an extra storage for future works.

In order to install Debian below steps are required:

1. Download the Debian latest image for the BeagleBone website. We need the version that did not flash the eMMC. This is the link: `http://beagleboard.org/latest-images`
2. Then follow these step from `http://beagleboard.org/getting-started` if you are using Windows. Else, if you are using Linux take below steps:
   (a) Uncompress the downloaded file with `unxz image_file.img.xz`
   (b) Locate the SDcard with `fdisk -l`. If you are using microSD adapter, then it could be linked as `/dev/mmcblk` or if you are using USB-SD converter, the device name might be linked as `/dev/sdb`

(c) Make sure the SDcard is not mounted, you can use `mount -a` or `umount /dev/{sdb or mmcblk}`

(d) Then you can load the image to SDcard with:
`dd if=imagefile.img of=/dev/sdb bs=4M conv=fsync`

3. Then insert the SDcard into the BeagleBone Black and hold down the boot button, then turn on the BeagleBone Black. The button need to be held down to make BeagleBone Black boot from microSD card.

4. Because we used a 4GB image for our 8GB microSD card, the file system that we boot from is 4GB. Therefore, we need to resize the partitions in order to use the extra space on the microSD card. In order to do that follow the steps in following link: `http://elinux.org/Beagleboard:Expanding_File_System_Partition_On_A_microSD`.

4.1.2 Accessing BeagleBone Black

In order to access BeagleBone Black via terminal when you are connected via USB to it you can SSH to `192.168.7.2`.

You can bring up a graphical interface using VNC protocol. In order to do first SSH to the BeagleBone Black and follow belows steps:

1. Install a VNC client on you machine. You can use `vinagre` which is the default on Ubuntu.

2. Install `x11vnc` on you BeagleBone Black with (if you are not connected to the Internet first look at 4.1.3:
`sudo apt-get install x11vnc`

3. Start the VNC server on BeagleBone Black with below command:

```
x11vnc -bg -o %HOME/.x11vnc.log.%VNCDISPLAY -
    ↪ auth /var/run/lightdm/root/:0 -forever
```

4. Now type `vinagre 192.168.7.2:5900` on you Ubuntu's terminal to start the remote desktop. You can use other clients with the address provided. This address is when you are connecting to the BeagleBone Black with the USB.

4.1.3 Configuring Internet

For connecting to the Internet there are two ways:

- Connecting the BeagleBone Black to an Ethernet cable
- Use Internet over USB

For using Internet Connection on BeagleBone Black, we can easily connect it to an ethernet cable. However, if want to access to the BeagleBone Black we need its IP. Since the IP is given dynamically with DHCP if we want to access it we are going to need its new IP each time we connected it to the network. To solve this problem we need to force BeagleBone Black to take a static IP from the network. We need to be careful that we do not assign another active IP to the BeagleBone Black, because this will lead to contentions in the Network. For the rest of the steps please follow this link `http://www.howtoforge.com/debian-static-ip-address`.

To simply use Internet we you are connected to the BeagleBone Black via USB you need to configure a new Ethernet connection to it. Since, we use `192.168.7.2` for SSH purposes we use `192.168.7.1` for Internet Connection. To configure the connection follow below steps:

1. Configure you machine for a new Ethernet connection:
   `ifconfig eth1 192.168.7.1`
2. On BeagleBone Black use these commands:
   `/sbin/route add default gw 192.168.7.1`
   `echo "nameserver 8.8.8.8" >> /etc/resolv.conf`
3. On your local machine enter these commands:
   `sudo iptables -A POSTROUTING -t nat -j MASQUERADE`
   `echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward > /dev/null`

These steps will configure the Interet over USB for the BeagleBone Black in the current connection. If you restart the BeagleBone Black you will need to do these steps again.

4.1.4 OpenHAB Configuration

To make openHAB run on the BeagleBone Black we need to install Java on it. We can use both Oracle's or OpenJDK. (on this project we have used OpenJDK)

For Oracle's version do following steps:

```
\item \verb+sudo apt-get update && sudo apt-get
    ↪ install oracle-java7-jdk+
\item \verb+sudo update-java-alternatives -s jdk
    ↪ -7-oracle-armhf+
```

For OpenJDK we need version 6, so use below command to install it:
`apt-get install openjdk-7-jre`

4.1.5 HABmin Configuration

The process of running HABmin along openHAB is straightforward:

1. Download the project zip file from GitHub and unzip files in the directory `webapps/habmin` (you will need to create this directory - note that the directory name must be lower case).

2. Place the org.openhab.io.habmin*.jar file into the add-ons directory (this is stored in the add-ons directory in the repository).

3. Place the org.openhab.binding.zwave*.jar into the add-ons directory (this is stored in the add-ons directory in the repository). Note that this bundle is currently required for HABmin to start, but if you don't have zwave then it won't actually run if it's not configured. In the longer term this dependency will be removed.

4.2 OpenHAB

Here we describe opanHAB configuration in details. First, we investigate items, they could be some parts of real sensors or virtual variables. Then, we talk about webapps, a very useful plug-in for openHAB. Then, writing the rules for different sensors and states are examined. Finally, we describe database management and Z-wave binding for connecting with Z-wave devices.

4.2.1 Items

Items provides an abstraction to manage sensors, and store their current state. For this reason openHAB also offers the Items Repository, which is connected to the Event Bus and keeps track of the current status of all items (See Figure 5). The Items Repository can be used whenever it is necessary to access to the current states of items. The Items Repository pervent each of the bundles to cache states themselves for there internal use. It also makes sure that the states is in sync between bundles. Also, it provides the possibility to persist states to the file system or a database, so they can be kept even on a system restart.

The items have some very useful characteristics:

- Items are objects that can be read from or written to in order to interact with them.
- Items can be bound to bindings i.e. for reading the status from e.g. KNX or for updating them.
- Items can be defined in files in below folder: `${openhab_home}/configurations/items`.
- All items definitions files must have the file extension `.items`

Items are defined in the followng syntax:

```
itemtype itemname ["labeltext"] [<iconname>] [(group1,
    ↪ group2, ...)] [{bindingconfig}]
```

where:

itemtype it is the type of the item. It defines the type of the interactions that can be used. Examples of itemtype are: Color, Contact, Dimmer, Group, Number, Rollershutter, String and Switch

itemname it is the unique name of the object which can be used in other files as well. For instance, it is going to be used with this name in the rules and sitemap files.

labeltext it is used on the sitemap to display a description for a specific item to user.

iconname The icon name is used to reference a png image file from folder `${openhab_home}/webapps/images/`

groups Items can be linked to specific groups by referencing groups by their names in a comma separated list embraced by round brackets.

bindingconfig Items can be bound to specific openHAB bindings by adding a binding definition in curly brackets at the end of the item definition.

One example of an item would be:

```
Contact Motion_Sensor_Bedroom1 "Bedroom 1 Sensor [MAP(
    ↪ motion.map):%s]" (Zwave,Bedroom1) { zwave="15:0:
    ↪ command=sensor_binary,respond_to_basic=true" }
```

4.2.2 Webapps (sitemap)

Sitemaps are used to create elements of an user interface for making openHAB items accessible to various front-ends.

Sitemaps can be composed by grouping various user interface elements into areas, which will be rendered by openHAB.

The basic structure of the sitemaps is:

```
#Obligatory section
[sitemap]
#Optional sections
[Colorpicker]
[Chart]
[Frame] #Area with either various other sitemap elements
    ↪ or further nested frames
[Group] #Renders all elements of a given group defined in
    ↪  an items definiton file
[Image] #Renders an image
[List]
[Selection]      #Gives access to a new page where the
    ↪ user can choose among values defined in the
    ↪ mappings parameter.
[Setpoint]       #Shows a value and allows the user to
    ↪ modify it. Optionally, it is possible to specify
    ↪ maximum and minimum allowed values, as well as a
    ↪ step.
```

```
[Slider]          #Renders a slider
[Switch]          #Renders a switch item
[Text]  #Renders a text element
[Video] #Displays a video
```

The sitemap element is mandatory in a sitemap definition file, its structure is:

```
sitemap [sitemapname] [label="<title of the main screen>"
    ↪ ]
```

For this project the most important optional sections are: frame, group, switch and text. (The chart is explained in section 4.2.4) In below we describe each part separately.

4.2.2.1 Frame

Area with either various other sitemap elements or further nested frames. It is the main grouping structure inside a sitemap. Frames are used to create a visually separated area of items.

```
Frame [label="<labelname>"] [icon="<icon>"] [item=<item>]
{
    [other sitemap elements]
}
```

4.2.2.2 Group

A Group element creates a click-able area that opens up on a new page, where you can show various elements. The sintax is:

```
Group [item=<itemname>] [label="<labelname>"] [icon="<
    ↪ iconname>"]
```

4.2.2.3 Switch

Renders a switch item. The syntax is:

```
Switch item=<itemname> [label="<labelname>"] [icon="<
    ↪ iconname>"] [mappings="<mapping definition>"]
```

4.2.2.4 Text

Renders a text element. The syntax is:

```
Switch item=<itemname> [label="<labelname>"] [icon="<
    ↪ iconname>"] [mappings="<mapping definition>"]
```

4.2.2.5 Dynamic Sitemaps

Depending of the value of certain states, we can choose to not show some frames into the sitemap, or add colors depending on their state, or the state of another item. A few use cases for this are:

- Hide elements depending on its mode
- Display different charts or URLs depending on the state of an item
- Show a battery warning if the voltage is low
- Highlight a value with a warning color if it's above or below limits

All widgets in the sitemap have the following three parameters available:

- visibility
- label color
- value color

Visibility

To dynamically show or hide an item, the visibility tag is used. By default, an item is visible if the visibility tag is not provided.

The format of the visibility tag is as follows:

```
visibility=[item_name operator value, ... ]
```

Some examples are:

```
visibility=[Weather_Chart_Period!=ON]
visibility=[Weather_Chart_Period=="Today"]
```

Multiple rules can be provided using a comma as a delimiter. Valid operators are the ==, ¿=, ¡=, !=, ¿, ¡.

4.2.3 Rules

Using the states of the previously defined items, or by defining periodic timestamps, we can use the Rules integrated into openHAB to execute actions.

Rules are placed in the folder `${openhab.home}/configurations/rules`.

A rule file can contain multiple rules. All rules of a file share a common execution context; they can access and exchange variables with each other. It is better to create a new file for every different context.

A rule file is a text file with the following structure:

```
[Imports]

[Variable Declarations]

[Rules]
```

Imports section contains files and descriptions that are going to be included into the rules.

The variable declarations section can be used to declare variables that should be accessible to all rules in this file.

An example for the variables sections would be:

```
var shared_variable = 0
val msg = "String container"
```

The Rules section contains a list of rules. Each rule has the following syntax:

```
rule "rule name"
when
    <CONDITION1> or
    <CONDITION2> or
    ...
then
    <ACTION_TO_EXECUTE>
end
```

A rule can have any number of trigger conditions, but must at least have one. When the condition are met, then the script on *action to execute* is executed.

The type of conditions for the rules are::

Item(-Event)-based triggers: They react on events on the openHAB event bus (updates or changes to items). Example to this type of conditions are:

```
Item <item> received command [<command>]
Item <item> received update [<state>]
Item <item> changed [from <state>] [to <state>]
```

Time-based triggers: They react at predefined times. Examples of this triggers are:

```
Time is midnight
Time is noon
Time cron "<cron expression>"
```

System-based triggers: They react on certain system statuses, such as:

```
System started
System shuts down
```

Besides the implicitly available variables for items and commands/states , rules can have additional pre-defined variables, depending on the trigger that occurs:

- Every rule that has at least one command event trigger, will have the variable receivedCommand, which can be used inside the execution block.
- Every rule that has at least one status change event trigger, will have the variable previousState available, which can be used inside the execution block.

The ¡action to execute¿ section is a *script* as defined in the openHAB documentation. The basic structure is the same used for Xtend language, for more information use the reference [17]

On the script we have access to all defined items,such as their name, all enumerated states/commands, e.g. ON, OFF, DOWN, INCREASE etc. and all standard actions to make something happen. An example of this would be:

```
if(Motion_Sensor_Hall.state == CLOSED){
        sendCommand(ZHallway, OFF)
}
else{
        sendCommand(ZHallway,ON)
}
```

4.2.4 Persistence (Database Management)

The persistence is intended to store item states over time (a time series). openHAB is not restricted to a single data store. Different stores can co-exist and configured independently.

There are many type of storage and databases that are supported by openHAB, like: MongoDB, MySQL, RRD4J, but we are only to focus on the last two.

4.2.4.1 MySQL Log

- Create a file in /configurations/persistence named mysql.persist
- This persistence service can be configured in the "SQL Persistence Service" section in openhab.cfg. The required items are:

```
mysql:url=jdbc:mysql://localhost/openhab
mysql:user=<user>
mysql:password=<password>
```

- Create database using mysql terminal

```
create database <DBNAME>;
grant usage on *.* to <DBUSER>@localhost identified
    ↪ by '<DBPASS>';
grant all privileges on <DBNAME>.* to <DBNAME>
    ↪ @localhost;
```

Fill the file mysql.persist with a file containing the rules.

The rules for the mysql persistence are intuitive and uses a CRON-like time schedule. An example for a arduino button is shown next:

```
Strategies {
        everyHour : "0 0 * * * ?"
        everyDay : "0 0 0 * * ?"
        everyMinute : "*/1 * * * * ?"
        default = everyChange,everyUpdate,
            ↪ restoreOnStartup
}

Items {
        ArduinoButton : strategy = everyChange,
            ↪ everyUpdate,everyMinute
}
```

The scheme used to store this information in the database is really simple. For each item that is being store it creates a new entry into the "Items" Table, then using the index stored in this table it creates a table such as ItemIdx (Item1, Item2, etc) to store the information related to that item. This tables contains all the information of the item, stored as a timestamp and value..

4.2.4.2 RRD4J Charts and logs

This type of log is really useful to show real-time graphs of the state of an item.

A good example of how to configure RRD4J is included in the main demo for OpenHAB and in the webpage [14]

To store the equivalent to the previous example of the Arduino, but using RRD4J, we change the file rrd4j.persist as follows:

```
Strategies {
        everyMinute : "0 * * * * ?"
}
Items {
        ArduinoButton : strategy =  everyMinute,
            ↪ restoreOnStartup
        Temperature*,Weather_Chart* : strategy =
            ↪ everyMinute,
    restoreOnStartup
```

```
}
```

To configure a chart using the previous store information of the button. The next line has to be added to the sitemap:

```
Frame label="Charts"{
    Text label="Test"{
            Frame{
                Chart item=ArduinoButton period=h refresh
                    ↪ =600
            }
    }
}
```

4.2.4.3 Persistence configuration

Depending of the type of sensor described we store the information in different type of databases, for the actual information of the sensors, described in the code as $C < Sensor >\_< Position >$ or $Motion\_Sensor\_< position >$ are stored only into the MySQL database, because we are not interested in graphing this information. This can be done as shown before.

In the current iteration of OpenHAB the charts can only be created using information coming from a rrdj4 database, then for all the variables related to Activity, named as Activity or Activity¡Position¿, are going to be stored in MySQL and rrdj4. The rrdj4 are only used to show the charts in the same way as described in the previous subsection.

We also add a cron rule to the rules, to enforce consistency every five minutes, as shown in the next code:

```
/** Hallway Flag */
rule "Hallway Flag activity"
when
        Item flag_hallway changed from OFF to ON or
        Time cron "*/5 * * * * ?"
then
    <Action>
end
```

4.2.5 Zwave Binding

The OpenHAB Z-Wave binding allows you to connect to your Z-Wave wireless mesh network. A Z-Wave network typically consists of one primary controller stick, zero or more additional controllers and zero or more Z-Wave enabled devices, e.g. dimmers, switches, sensors etc. Connection to the Z-Wave controller is done through the serial port of your host system. USB controllers typically create a virtual COM port to connect to the stick. Please write down the port

name before configuring this binding. In case your port name changes dynamically and you want to use a symlink, see Tricks. A list of supported controllers is listed below.

Initialization of the binding typically takes several seconds to minutes depending on the number of devices in the network. When battery operated devices are used the binding tries to reach the device first. After one minute the node is marked sleeping. On wake up of the device initialization will continue.

Before using this binding is better to check all Z-wave devices is functioning. For configuring this binding HABmin can be used. First of we need to introduce the port settings of the Z-Wave controller in the openhab.cfg file (in the folder `${openhab_home}/configurations`). The variable is `zwave:port`. On our system the value is `/dev/ttyUSB0`. You can find the correct name in your system by looking at `/dev` directory. On important point before running the openHAB-runtime is to change the ownership of the device. This is done by the `chown $USER /dev/ttyUSB0`

4.3 Zwave Devices

Configuring Z-Wave products can be a little cumbersome when using OpenHAB, we need to use an additional application to configure the network of z-wave hardware. The best currently available option is to use HABmin.

In the current version of HabMin the process of configuring is completely manual (the automatic detection and configuration is in development). But still the Z-Wave configuration page allows reading and writing of node configuration parameters, association groups, wakeup interval etc. It is designed to support manual configuration of Z-Wave devices without having to stop openHAB.

HABmin uses a Z-Wave Product Database to allow configuration of the parameters and associations. The database is linked into the Z-Wave binding and needs to be updated for each device in order that HABmin will work correctly with that device.

Initially the interface will just list the nodes and if the node type is found in the z-wave product database, it will list the manufacturer and product types for all known nodes. The *parameters* under a node will show the parameters for a node.

The small indicator next to the device name can be gray, yellow, green or red. If it's grey, then it indicates that the device has not completed its initialization. If red, the node is DEAD, and if green, it's ALIVE. If it's yellow, then the node is currently operating ok, however it has been dead in the past 24 hours.

If the manufacturer and/or product aren't known, then the interface will list the manufacturer ID, device ID and device type. This will allow these devices to be added to the configuration database.

There are two main sections that has to be configure before any z-wave device is working: *associations* and *configuration parameters*. The associations permit a sensor to talk directly to other peripheral without requiring to go through the hub. The configuration parameters define the information, frequency and all

other configuration files that permit transmiting information from the sensor to the central hub. To configure the associations follow the next steps:

- Expand the Associations Groups in HABmin for the device you want to configure (ie the sending node)
- Expand the group that you want to modify. You will be presented with a list of all the nodes in your network.
- Change the state of the node that you want to receive the command from Non-Member to Member

Sometimes the sensors resist to configure any kind of association, so it is required to verify that the space turns from yellow to white and that the new modify value had changed, otherwise, repeat all the steps again. As a recommendation always associate each sensor to the Z-Wave Stick, so it sends information at any moment.

The configuration parameters are specific to each device, so this are described in more detail in the next sections.

4.3.1 Configuring the Z-Stick 2E

The Z-Stick operates in three distinct modes: Inclusion-Mode, Removal-Mode and SerialAPI-Mode. Both Inclusion-Mode and Removal-Mode require the Z-Stick to be unplugged from the USB connector of the host, while Serial API-Mode requires that the Z-Stick to be plugged into the USB connector of the host. All the inclusion of sensors are done using this device.

4.3.1.1 Adding a new device to the network

1. To initiate Inclusion-Mode, unplug the Z-Stick from the USB connector and then tap the button. (The LED will blink slowly.)
2. To include a new Z-Wave device into the network, simply go to the device with the ZStick and press the button on the device you wish to include. (The LED on the Z-Stick will blink fast during a network neighbor discovery and stay solid for 3 seconds to indicate successful inclusion of the device into the network.)
3. The LED will then return to blinking slowly, indicating readiness for further device inclusions. Repeat step 2 for each device as you wish to include.
4. Tap the Z-Stick button to turn it off.

4.3.1.2 Remove a device from the network

1. To initiate Removal-Mode, unplug the Z-Stick from the USB connector. Then press and hold down the button for approximately 3 seconds. (The LED will transition from blinking slowly to blinking fast.)

2. To remove a Z-Wave device from the network, simply go to the device with the Z-Stick and press the button on the device you wish to remove. (The LED on the Z-Stick will immediately stay solid for 3 seconds to indicate successful removal from the network.)

3. The LED will then return to blinking fast, indicating readiness for further device exclusions. Repeat step 2 for each device as you wish to exclude.

4.4 Commercial Sensors

Currently, different Companies produce Sensors for smart homes. We have used some of these sensors in our product. The sensors that we have used in our project use Z-wave protocol to connect to the Z-Stick on our central Unit.

There are some commercial Central Units too, like Wink[2] which can connect different product from different brands like Quirky, Nest and Philips and control them with a single app. It supports Z-wave, ZigBee and Bluetooth. However it does not support all brands, therefore our solution is still cover more sensors. On the other hand Wink is the first commercial product that connects different brands into one central unit. Figure 6 shows a simple representation of their system from their website.

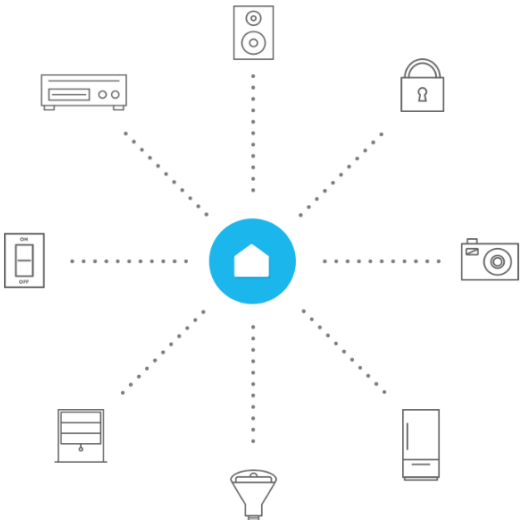

Figure 6: Wink System

We have used some commercial sensors in our project. These products use Z-wave technology for communication. We have used multi-sensors that include motion, humidity, temperature, light intensity and battery, door and window sensor, and also lamp modules as our actuators in different events. We included

[2] `http://www.wink.com/`

the price of these devices as an indicator of price range in these kinds of home sensors.

4.4.1 AEOTEC MultiSensor

The AEOTEC MultiSensor by AEON labs is a group of Sensors that is implemented in an motions sensor's main body. It includes motion, humidity, temperature and light (and device's battery) sensors. This sensor can be used indoor and outdoor. This price of this device is $60.

4.4.2 HSM100 MultiSensor

HSM100 is a multisensor by HomeSeer like the previous sensor without a humidity sensor. The price of this device is around $80.

4.4.3 AEOTEC Labs Door/Window Sensor

The Door & Window Sensor finds out something very simple - whether a door or a window, or in fact any object in your home, is open or closed. But theres power in simplicity. Power that allows you to inform the rest of your Z-Wave network about your selected doors and windows."[3]

This is simple Z-wave sensor that can be associated to other sensors and actuators. The association is a part of the Z-wave communication protocol and therefore each Z-wave sensors is able to support this. The price of this device is around $35.

4.4.4 Evolve & Jasco Lamp Module

We have used lamp modules that is controllable via Z-wave as our actuators. These modules simply connect and disconnect the power through their outlet via Z-wave commands or the button they have on their frame. These devices' price is around $55. In addition, because these modules have a direct power source they can increase the strength of the Z-wave mesh.

4.5 Custom Sensors

In this subsection we investigate our custom sensors.

4.5.1 Arduino-based sensors

We can use Ardunio as a base for collecting basic data and sending them to the central unit. In this project we use a series of Ardunios which are connected via ZigBee to the main Arduino that is connected to our central unit via a USB cable.

Following subsections focus on how we can configure this system.

[3]From the product website: `http://aeotec.com/z-wave-door-window-sensor`

4.5.1.1 Arduino and XBee in API Mode

The process to configure an Arduino Uno and Xbee can be long, but only have to be done once for each device. This is explained step by step next:

1. Install XCTU from [27]. For ubuntu you can use wine, but you need to configure the COM port correctly.
2. To configure the XCTU on the arduino UNO with the arduinoXbee v1.1 shield, follow the next steps:
   (a) Disconnect the ATMEGA microcontroller from the Arduino.
   (b) Change the jumpers in the shield (both) to the USB position (i.e. on the two pins nearest the edge on the board)
   (c) Connect the shield to the Arduino
   (d) Connect the Arduino to the computer using the USB cable.
3. Update the firmware of the xbee:
   (a) Download the firmware from [16], that matches the one on the Arduino. This in order to have the configuration files.
   (b) Configure the XBee as follows
      i. Open the X-CTU software and type the Arduino XBee module's COM port number into the "Com Port Number" field. Then click the Add button. The COM port you created will appear in the two text boxes on the left. Select both and then click the "Test / Query" button, will appear a screen saying that the communication to the device is OK, and must show the serial number and firmware version. If this does not works, you should change the configuration from the port in the operating system.
      ii. Go to the modem configuration tab, the settings that are most important are the PAN ID (ID)and the serial number high word (SH) and low word (SL). Radios which share a PAN ID can communicate with each other. Radios on different PAN IDs will ignore each others messages. The Serial number consists of two 16-bit numbers, stored in two addresses: SH (serial high) and SL (serial low). This is the radio's unique address, and is used to send messages to it. In the settings below you only need to change the PAN ID. Use the same number for both radios so that they can communicate. Read the SH and SL parameters and take note of them.
      iii. For the coordinator configure the device function set as *ZNET 2.5 COORDINATOR API 1147* and change the next values as:

```
First restore to factory settings to eliminate
    ↪ the chance of lingering settings *
RE
```

```
Set PAN ID to an arbitrary value. The end
    ↪ device must also use this exact value
ID=1AAA

Both radios must have the same Channel and PAN
    ↪ ID to communicate
CH=13

Set the node identifier to an arbitrary string
    ↪ . This is optional and only serves as a
    ↪ convenient way to identify your devices
NI=COORDINATOR

Set API mode to 2 (escape control bytes)
AP=2

Save to settings to survive power cycle
WR

Reboot the radio. apply changes "AC" should
    ↪ also suffice
FR
```

iv. For the router/end configure the device function set as *ZNET 2.5 ROUTER/ENDDEVICE API 1347*, the configuration of the Router/End Device is the same as the coordinator, except choose a different value for the Node Identifier, ex:

```
NI=END_DEVICE_1
```

v. Check the "Always Update Firmware" in the check box and press "Write" button, and the configuration will be uploaded on your module

Most of the work is done in the Arduino Script, using an specialized library [25]. Look at the addendum for the actual code.''

To be able to read the information coming from the coordinator, we need to use a second Arduino, using the software serial library (¡SoftwareSerial.h¿) on both side, in the reader and the coordinator. Then using the SoftwareSerialReader file in the reader Arduino and connect the ground of both tougher, and use the pin 8 for the reading device connected to the pin 9 of the other Arduinos. Use this code to display on the serial monitor of the Arduino IDE.

Finally we need a third Arduino to use as a software serial port to read what is happening in the other Arduinos, remember to connect the ground, and use the pin 8 for reading connect to the pin 9 of the other Arduinos. Use this code to display on the serial monitor of the Arduino IDE.

4.5.1.2 Arduinos Communication Protocol

Because the maximum information that the Arduino can retrieve from the ADC converter is 10-bits (Arduino resolution). Then sending information from one Arduino to another using API ZigBee (see here for configuration), we are going to use the first 10 bits for information and the remaining 6 bits for node identification (64 nodes), if more nodes are required, we could send a payload package of 3 members, instead of two. Each value in the payload has a size of 8 bits. For the project we assume a payload of size two, each of size 8 bits.

Using this convention then we have this new codes for the Arduinos. Remember to change the destination number (obtained from the configuration step) and the node number for each node to identify itself.

4.5.1.3 Information about the source code attached

All the source code for the Arduino Sensors, are ready to use, the only section that initially required to be modified are the constants at the beginning of the code, that define values such as:

NODE NUMBER : Is the node number that the coordinator is going to see when this Arduino send information, be sure to write all the node numbers different, otherwise the OpenHAB is not going to be able to differentiate them.

SH : Is the serial number high word from the coordinator, it defines to whom the Arduino is talking.

SL : Is the serial number low word from the coordinator, it defines to whom the Arduino is talking.

DELAY : defines the distance in time between sensing values send to the coordinator.

READS DELAY : defines the distance between reading of the actual sensor, it is different from the previous one in that this is that this is how fast we sense the environment, the previous one is how much data we send to coordinator, the previous value is to reduce the network requirements.

PIN NUMBERS : this includes descriptions such as VCC, GND, ECHO, TRIG, that defines the pins used in the actual board. The default configuration for normal sensors, use the A0 pin to read the information, pin 12 as VCC and the actual GND of the Arduino as GND. All of this except the GND can be changed in all the Arduino scripts included in this project.

4.5.1.4 Ultrasonic sensor

This is a 4 pin sensor, 2 of them are used as Voltage Inputs (VCC and GND) and the other two are the actual information coming from the sensor, named ECHO and TRIGGER. The process of reading the information is quite simple,

we send a pulse signal with at least 2 ms in high, and then we read the pulse coming from the echo. Then we can use a simple function, measuring the time and using the velocity of sound. An extract of the code is shown here (the complete source is included in the source code attached)

```
// The PING))) is triggered by a HIGH pulse of 2 or
    ↪ more microseconds.
// Give a short LOW pulse beforehand to ensure a clean
    ↪ HIGH pulse:
digitalWrite(TRIG, LOW);
delayMicroseconds(2);
digitalWrite(TRIG, HIGH);
delayMicroseconds(5);
digitalWrite(TRIG, LOW);

// The same pin is used to read the signal from the
    ↪ PING))): a HIGH
// pulse whose duration is the time (in microseconds)
    ↪ from the sending
// of the ping to the reception of its echo off of an
    ↪ object.
duration = pulseIn(ECHO, HIGH);

// convert the time into a distance
cm = microsecondsToCentimeters(duration);
```

For this sensor all the pin positions can be configure in the definition section of the Arduino script with the names: VCC, GND, TRIG and ECHO.

4.5.1.5 Pressure sensor

The pressure sensor is actually a resistor controlled by the force applied to him, because the sensor measure the force applied in an area, but actually the sensor does not differentiate the actual area being used. For this configuration we use a voltage divisor to measure the changes, we initially match the resistance of the sensor, without any pressure measured, when the voltage drops 100 units (of 1024 being measured by the Arduino), then the Arduino send a signal to the coordinator indicating that activity was detected. As shown in figure 7, in this figure, we connect Vout to the A0 pin, Vin to the 12 pin and GND to the GND pin, using the code provided, this completes the actual sensor.

4.5.1.6 Door movement detection using IR

We have used infrared transmitter and receiver to detect motion through the doors. For the circuit we compare the voltage of the IR receiver with some voltage which is controllable with a potentiometer. The value of the voltage is dependent on the environment and the distance of the IR transmitter. Then we

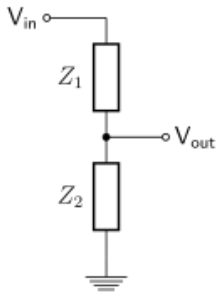

Figure 7: Voltage Divider

read the analog values and compare them. The rest of the implementation is like the rest of the Arduinos.

5 Performance Analysis

Based on the analysis of the openHAB architecture, we have defined the main performance point to be analyzed in future works:

5.1 Event Bus

This is one of the main aspects to be analyzed in relation to the performance. All the items are connected to the Event Bus, either doing a state update or receiving/sending a command. As can be shown on the image 8.

This bus can get crowded, so special attention could be presented to it.

5.2 Stateful Event Repository

It is the internal functionalities of OpenHAB, such as the item registry, automation rules, etc. Probably this is not going to be the main bottleneck of the system. Overall, it is going to be included in the analysis of CPU consumption, so it does not have to be analyzed separately.

5.3 Turn off and restart time

The server has to configured to work in unstable grid conditions, e.g. what happens it is turned off unexpectedly. It has to be able to reboot to a known state, and start working as fast as possible.

It is required to precisely determined what is the value of "as fast as possible", simulate a shutdown of the system, an notice if it is able to recognize all the systems connected to the network and be able to respond to any modification. What happens with the information that was not retrieved during the restart of the system, is it going to get lost

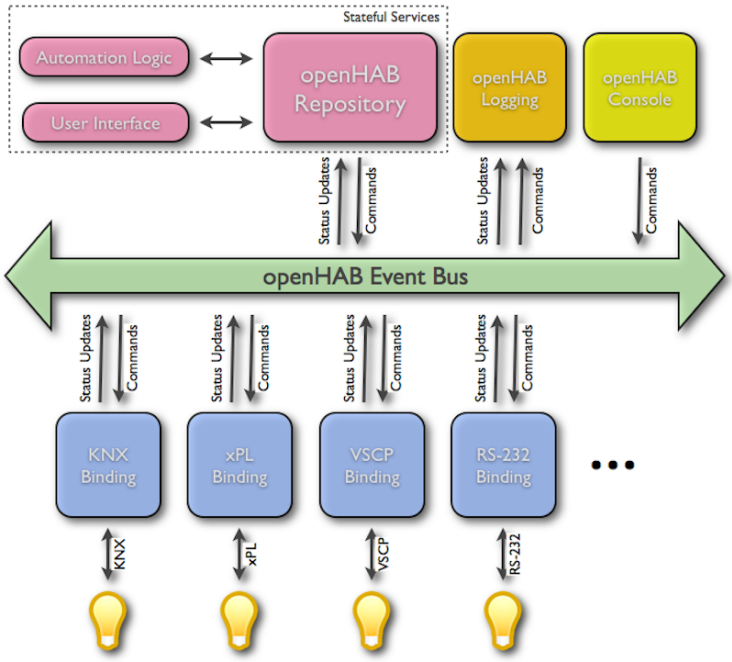

Figure 8: Event bus

5.4 Logs and Databases

Al this information is managed through the Persistence module, sending the information to the databases and other log services. We should analyze how often are this updates being done, and see if there is any true requirement to the network or the system (BeagleBone Black)

6 How Execute all

After configuring each device individually, writing codes for openHAB configuration and configuring central unit, the server can be run with:
`<openhab_server_dir>/start.sh`
Also, the server could be executed in the debug mode with:
`<openhab_server_dir>/start_debug.sh`
But be careful about the size of the log files in this case. After starting the server, Z-wave devices can be configured on a web with this address:
`http://<server_IP>:8080/habmin/index.html`
If you are accessing the web on your server use `localhost` or `192.168.0.1` instead of the IP. After each device is configured and is ready to collect the data, the main interface can be accessed via:
`http://<server_IP>:8080/openhab.app?sitemap=demo`

or `http://<server_IP>:8080/greent`.

For further demonstration of execution demo you can watch the video we have recorded for our demo (For higher quality download it and then play). You could access our video from here: Link
or here: Link2

7 Future Work

- We could reduce the usage of Arduinos by using XBee module for processing data. So, we are going to implement some low compute-based sensors only with XBee module.
- Currently, pressure sensors are detecting motion based on a point. We could extend this to a whole area by using multiple sensors with a single Arduino.
- A possible future work could be detecting patterns in activities and use them for useful computations such as Disease detection, activation of particular actuators and etc.
- We can use this framework for providing data to Doctors and Health centers. We need to implement communication layers in the openHAB.
- Make out system more reliable for detecting humans. Currently, everything that moves can trigger events.

8 Related Works

The related works is divided into two main subsections: software for the testbed and the system for the application of it.

8.1 AwareHome Base System

This system is equivalent to a Home Automation Software base, we need to be able to connect to multiple sensors, and be able to send information to actuators and emergency systems. There are various open source projects related to this application:

OpenHAB : *vendor and technology agnostic open source automation software for your home.* [9] This is a software interface that is vendor-neutral. It works as a Java Virtual Machine, so it can be use in the platform proposed in the next section, with the idea of being easily extensible. We use this platform as our base.

MisterHouse : *MisterHouse is an open source home automation program.* [7] It is a perl-based system, that trigger events based on time, web, socket, voice, and serial data. It seems to be a little outdated, with less support to newest protocols, but could have some utility to the related project.

Eclipse SmartHome : it is a Open-Source project initiated by the eclipse foundation, *The initial contribution to the Eclipse SmartHome project comes from the open source project open Home Automation Bus (openHAB)* [5], as the webpage states is an extension of the OpenHAB interface system. It is *a flexible framework for the smart home services, interfaces and tools for the "Intranet of Things"*. The system uses *bindings* to interact with other objects in the net; a binding is an extension to the Eclipse SmartHome runtime that integrates an external system like a service, a protocol or a single device. It seems to be a really attractive option to implement the testbed for the system.

MQTT : *is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport.* [8] It could be use in connection with the binding in eclipse SmartHome, to streamline the connections, an produce a testable interface for all the systems.

8.1.1 Invisible Monitoring System

There are various companies that are designing systems similar to the one presented in the previous section. They either offer a costly functional system or a very limited one. These systems present a base of proved sensors that are going to work in the proposed project. The available resources are separated in two main sections *Commercial Systems* and *Academic Research*.

8.1.2 Commercial Systems

- BeClose: Senior Safety System [12]
  The system has many of the capabilities of the intended project, but with a very restrictive price. The hardware cost for a complete system is about $ 575 and a recurrent monthly payment of at least $ 69. For most countries this is not affordable.

- HealthSense: eNeighbor Remote Monitoring System [18]
  Monitors and sensors combine to meet the specific health needs of a resident, both on campus and in the community. From managing acuity transition points to providing early memory care assistance, eNeighbor helps communities meet care, occupancy and resident satisfaction goals across the continuum of care[18]. This system is interesting for the project because it presents an scalable possibility of taking care of multiple elderly patients at the same time. The caregivers are able to prioritize their actions in relation to the actual needs of each person.

- WellAware Systems [19]
  ... offers a patented, integrated solution comprised of unobtrusive, highly accurate sensors and software, enabling professional caregivers and nurses to gather and analyze critical wellness trends ... [19]. Integration of sensors is one of the main objectives of this project. The WellAware Systems presents an example of how basic sensors could be integrated to give responses that are readable by men.

- GrandCare Systems [28]
  In this system wireless health devices, interactive assessments, and medication management tools is measuring the person wellness. Also, activity sensors monitor daily activities without impeding on the person's independence or privacy. Additionally to the previous systems, the person can instantly access video chat, photos, letters, and calendar events from family and caregivers on tablet (social networks). They also have a comprehensive system for family members and caregivers access through web pages and mobile apps. This system is more complex than the scope of this project.

There are plenty of more systems, but they are either too intrusive or poorly designed. Some references are: Lorex Live Solutions [15], Philips Lifeline [24], ADT Elder Care [11].

8.2 Academic Research

The next are the articles of investigations related to monitoring systems for elderly people.

- Aware Home Research Initiative [29]
  Aware Home Research Initiative (AHRI) at Georgia Institute of Technology is an interdisciplinary research endeavor involving faculty and student researchers from multiple domains across campus. AHRI researchers are interested in three main research areas: Health and Well-being, Digital Media and Entertainment, and Sustainability, investigating how new technologies can impact the lives of people at home.[29]. The web page does not have enough information about the current research articles and presentations. It presents a close-source of information for the analysis of the sensors that is used.

- Activity monitoring system for elderly in a context of smart home[13]
  This papers presents a system architecture and technologies required to create *a new multi-sensor monitoring system for the elderly with cognitive disabilities in a care unit*[13].

- Contempo: A Home Care Model to Enhance the Wellbeing of Elderly People.[23]
  This papers creates a model for measuring the wellbeing of elderly people,

and also design a pervasive health care system and an ambient environment. It is useful for the present project in the layers of connections used and the quantitative measure created.

- Altcare: Safe living for elderly people [26]
  This paper gives an overview of the project Altcare for providing an ambient assistant living environment based on visual activity monitoring. The present project is not interested in using complex sensors, such as cameras, but the pipeline for the stages in the detection of activities could be useful for integrating the information in the sensors.

- Wireless, Multipurpose In-Home Health Monitoring Platform: Two Case Trials[22]
  The monitoring platform is a multipurpose system which is easily configurable for various user needs and is easy to set up. It present a new market strategy where the system could be temporarily rented from a service company by, for example, hospitals, elderly service providers, specialized physiological rehabilitation centers, or individuals. It also presents an analysis of the correct way to select the wireless protocol for a fully functional system.

- Virtual Caregiver: An Ambient-Aware Elderly Monitoring System[21]
  They develop a system that captures the ambient context of the environment including the elderly activities of daily living, and based on the captured context it dynamically determines whether to provide implicitly services to the elderly or call for additional support. It presents a data flow graph that explains the interaction between the sensors, the situation analyzer system and the actual service invocation manager. It also express the grammar for the creation of rules for the system.

- A Human Caregiver Support System in Elderly Monitoring Facility [20]
  The paper introduces human caregiver support system in the context of elderly monitoring, which works by considering the surrounding context and activities of the elderly people. Second, it presents different modes of interaction of the human caregiver support system for monitoring effective the elderly and reducing the cognitive load of the human caregiver. Its main focus is on the caregiver (family), and how to reduce the strain of taking care of elderly people.

References

[1] Announcement: The winner of the iot challenge 2013!! `http://iotevent.eu/application-2/announcement-the-winner-of-the-iot-challenge-2013/`, 2014.

[2] Beaglebone. `http://beagleboard.org/`, 2014.

[3] Do we need an intranet of things? http://vasters.com/clemensv/2014/02/14/Do+We+Need+An+Intranet+Of+Things.aspx, 2014.

[4] Draft:openhab. http://en.wikipedia.org/wiki/Draft:OpenHAB#cite_note-2, 2014.

[5] Eclipse smarthome. www.eclipse.org/smarthome/, 2014.

[6] Java. https://www.java.com/en/, 2014.

[7] Misterhouse. http://misterhouse.sourceforge.net/, 2014.

[8] Mqtt. http://mqtt.org/, 2014.

[9] Openhab. http://www.openhab.org/, 2014.

[10] Raspberry pi. www.raspberrypi.org, 2014.

[11] ADT. Elder care. http://www.adt.com/senior-safety, September 2014.

[12] BeClose. Senior safety system. http://beclose.com/beclosesystem.aspx, September 2014.

[13] Y. Charlona, W. Bourennanea, F. Bettahara, and E. Campoa. Activity monitoring system for elderly in a context of smart home. *Digital technologies for healthcare*, 34:60–63, February 2013.

[14] The Embedded Code. Persistance example. Click Here for Link, August 2014.

[15] Flir Company. Lorex live solutions. http://www.lorextechnology.com/solutions/Elderly-Monitor-solutions/2200031.s, September 2014.

[16] Digi. Xbee firmware. ftp://ftp1.digi.com/support/firmware/, November 2014.

[17] Eclipse. Xtend documentation. http://www.eclipse.org/xtend/documentation.html#Xtend_Expressions, November 2014.

[18] HealthSense. eneighbor. http://www.healthsense.com/index.php/products/remote-monitoring/eneighbor, September 2014.

[19] HealthSense. Wellaware. http://www.healthsense.com/index.php/products/remote-monitoring/wellaware-systems, September 2014.

[20] M.A. Hossain and D.T. Ahmed. A human caregiver support system in elderly monitoring facility. *Multimedia and Expo Workshops (ICMEW), 2012 IEEE International Conference on*, 2012.

[21] M.A. Hossain and D.T. Ahmed. Virtual caregiver: An ambient-aware elderly monitoring system. *qInformation Technology in Biomedicine, IEEE Transactions on*, 2012.

[22] S. Junnila, Tampere, H. Kailanto, J. Merilahti, A. Vainio, and more authors. Wireless, multipurpose in-home health monitoring platform: Two case trials. *Information Technology in Biomedicine, IEEE Transactions on*, pages 447 – 455, 2010.

[23] Kurnianingsih, Lukito Edi Nugroho, Widyawan, Lutfan Lazuardi, Ridi Ferdiana, and Selo. Contempo: A home care model to enhance the wellbeing of elderly people. *Biomedical and Health Informatics (BHI), 2014 IEEE-EMBS International Conference on*, pages 472 – 475, 2014.

[24] Philips. Lifeline. `http://www.lifelinesys.com/content/`, September 2014.

[25] Andrew Rapp. Arduino-xbee library. `https://code.google.com/p/xbee-arduino/`, November 2014.

[26] Muhammad Shoaib, Tobias Elbrandt, R. Dragon, and J. Ostermann. Altcare: Safe living for elderly people. *Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2010 4th International Conference on-NO PERMISSIONS*, pages 1 – 4, 2010.

[27] DigiXCTU Software. Xctu software. `http://www.digi.com/support/productdetail?pid=3352&type=utilities`, November 2014.

[28] GrandCare Systems. Grandcare. `http://www.grandcare.com/`, September 2014.

[29] Georgia Tech. Aware home research initiative. `http://www.awarehome.gatech.edu/`, 2014.