"Starter kit for smart buildings"

Pervasive Computing : Selected Topics Project

Aebischer Nadia Department of Informatics
University of Fribourg
1700 Fribourg, Switzerland
nadia.aebischer@unifr.ch

Luyet Gil Department of Informatics
University of Fribourg
1700 Fribourg, Switzerland
gil.luyet@unifr.ch

ABSTRACT

The goal of this report is to describe an easy way to integrate a basic pervasive system into an already built house/building. The system should be cheap and easy enough to integrate and to operate so it could be used by the masses as a starter kit for smart buildings. The report will be based on the data analysis and the context of use of this system. Especially about the central unit managing the whole house.

1. INTRODUCTION

1.1 Smart building description

A smart building is by definition a building enhanced by the technology that is managing it. Generally the technology used is splitted into four parts. First, we have the sensors that collect data and allow to have an understanding on what is going on outside or within the house. The second part is the network that allows to spread the data collected by the sensors and to eventually activate the actuators if needed. The third part is the managing unit that allows to gather data and infer knowledge about the environment and thus either show the data to the user or activate actuators to change the user's environment. The last part is the data visualisation that allows the user to interact with the data. This structure follows the given standard loop of a pervasive system shown in Figure 1.

1.2 Aim of the project

The basic aim of the project is simple. We would like to produce a simple to use and to integrate system that would allow to enhance an already built building to a smart building. This system should be relatively cheap and should involve the least possible physical changes in the house (e.g. destroying walls). This means that it has to be based on cheap and already existing means and be a sort of "do-it-yourself" kit.

2. HOUSE DESCRIPTION

2.1 Context of use

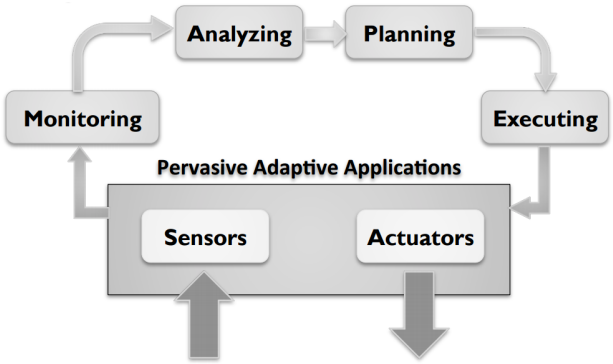

Figure 1: Pervasive system loop

We are going to consider a small building with a dozen of individual apartments. Each apartment would have four rooms, each one having one light and one window with an electric sunblind. In such a configuration it is usual to split the power supply into so call groups. Each group has its own electrical phase. For a standard four rooms apartment (one kitchen, three rooms and one bathroom) we would have three or four groups: one managing the kitchen, one managing the bathroom and one or two groups for the other rooms. The building has a central water heater, separated electric meters per apartment and an individual thermostat per apartment. A central electrical panel is available for the whole building. An indoor parking space in common with an electric door that is activated by a remote controller is also considered.

2.2 What could be automated

Sunblinds are opening or closing automatically by checking the sunlight and time. For example if it's sunny the sunblinds will be closing. The same way, at night, they should be closing to protect our privacy. The temperature of the apartment can also be automated. For example we can set the temperature we want to have in the apartment, and if the actual temperature is below or above the default one we set, the system will change the thermostat accordingly. We can also imagine a way to infer if somebody is going to be there soon or not and raise more or less fast the overall temperature. This inference can be computed according to the presence of people in the house by reading the presence sensors. For example, if somebody is never present during the

weekends then the central unit should over the time learn that fact and thus it will heat the apartment in order to saving energy. The lights can also be automated by detecting the presence of people in a room. RFID tags linked with cars could be used to open the parking space doors. Of course, these automated behaviors can be stopped by the user if he doesn't want it to be performed. To do so, the user can access the central unit via a web interface and change the behavior the way he want at all times.

3. A FEW WORDS ABOUT THE SENSORS

In this section, we will briefly talk about the different sensors that our Central Unit will deal with. We are not going into too much details as it is not our main subject for this project.

3.1 Sensors architecture

The communication between the components are achieved by WiFi or over the electrical network in case the WiFi is not reachable. Communication over the electrical network can achieve a very fast connection speed between the components, up to 1200 Mbits/s [1]. This means that theoretically preprocessing of sensors data is not needed but for utility reasons we will preprocess the data before reaching the Central Unit. Each sensor is connected to a light computer, typically a Raspberry Pi could work but we can imagine an even less powerful one for small tasks. This light computer will preprocess the raw sensor data and play the role of the interface between the sensor and the Central Unit. This interface is needed in case of multiple types of sensors. By changing the sensors, one could simply apply a patch to the small computer (even via the Central Unit) and resolve compatibility issues. This interface used is a simple HTTP based communication via POST with a specific XML-based body very similar to the SOAP protocol. This produce a simple Service-Oriented Architecture (SOA) to access the data. Here is an example:

```
POST address
  <MESSAGE>
    <TYPE>(alert|event|failure|do)</TYPE>
    <FROM>(sensor_id|central_unit)</FROM>
    <CONTENT/>
  </MESSAGE>
```

Where the `TYPE` denotes an alert, an event or even a failure for example. The `FROM` and `CONTENT` parts deals respectively with the author of the message and the actual meaning of the message. A typical proximity sensor message would look like the following:

```
POST central_unit
  <MESSAGE>
    <TYPE>event</TYPE>
    <FROM>sensor_id2345</FROM>
    <CONTENT>distance:'2.34m'</CONTENT/>
  </MESSAGE>
```

A typical message sent by the Central Unit to open a door would be:

```
POST address_of_door_actuator
  <MESSAGE>
    <TYPE>do</TYPE>
    <FROM>central_unit</FROM>
    <CONTENT>door[0]:'open'</CONTENT/>
  </MESSAGE>
```

3.2 Location

The locations of the sensors depend of the type of sensor. Here is the list of the sensors we are going to consider:

Luminosity sensor - outside, at each frontage of the house.

Presence sensors - around the lights and the doors.

Temperature sensors - inside/outside.

Energy consumption sensor - after the electric meter.

Water consumption sensor - before the water inlet.

Water temperature sensor - after the water heater.

RFID readers - for cars around the garage doors.

We also consider actuators, the list of them is given here:

Sunblinds actuators - to open and close sunblinds.

Light actuators - to switch the lights on and off.

Thermostat actuator - to change the house temperature.

Automatic garage door opener - to be linked with the RFID reader.

This part is not really important for the scope we choose to aim our report, but this should be at least mentioned because it lists which sensors we are going to consider and we are receiving data from in order to form a context, and also on which actuators the Central Unit can send message to change the environment.

4. CENTRAL UNIT

The Central Unit is the main part of this architecture and the scope of this report. This Central Unit depends on several topics as its own architecture, the sensors data coming in, the actuators it has to interact with its environment and of course its standard interface with the user.

4.1 Architecture of central unit

The architecture itself has no specific requirement. We can choose whatever we want, here we might focus on a standard Linux distribution as Ubuntu without any graphical environment as the only interaction we have will be via a web interface discussed later. Furthermore the computation needed is not too high as the preprocessing of the sensor is done on the sensor side with light computer as discussed earlier. The computer should also be running 24/7 and thus the computer has to have a low consumption. A light computer as a Raspberry Pi might be enough.

4.2 Sensor fusion

The sensors fusion is performed to produce a context. A context is the actual state of an entity such as a room or a person. In this report we need to know exactly which kind of contexts we are going to consider to know exactly which sensor data has to be aggregated to produce the actual state of a room for example. The aggregation itself will be done in such a way that sensors failure could be minimized. A standard way to do this can be done via computing a mean value of the last dozen of incoming values and use this mean as the incoming sensors value.

4.2.1 Sensors message synchronization

Has some sensors output can be asynchronous, e.g. if we take the presence sensor it could output each time it detect a presence. This could be solve by using some kind of synchronization protocol similar to the WiFi communication protocol. Each sensor we have is connected to a little computer used for preprocessing of sensor inputs, this allow us to synchronize the data flow by offering time windows management. This kind of management is usually done by either give a time frame by sensor or allow one big time frame to allow each sensor to send periodically during a certain amount of time. The second option will be taken here but we might introduce a little modification. The size and the exact beginning of this time frame is not fixed but we introduce here Ready-To-Send (RTS) messages similar to Wifi collision avoidance protocol. Each time the Central Unit has finished to infer all the contexts it send a message to all the sensors a RTS message to notify them that they can send their data. Such a message would look like the following.

```
POST address_of_sensor
  <MESSAGE>
    <TYPE>event</TYPE>
    <FROM>central_unit</FROM>
    <CONTENT>'RTS'</CONTENT/>
  </MESSAGE>
```

During the time between two RTS messages, sensors are able to aggregate their sensors value to give a mean value of them. As the sensors are now synchronized we need to infer contexts from their data.

4.2.2 Context inference

Inferring contexts depends on the states from several sensors. In this part of the report we list the contexts choosen and how to obtain them. A certain context may depend on several subcontext or states, e.g. the lightning state of a room may depend of the outsite lightning, the state of the lights of the room and the states of the sunblinds.

The lightning state of the room
This context is composed of distinct contexts, the lightning of the room and the outside environment **("dark" or "lighted")** and the sunblind states **("open" or "closed")**. Of course those contexts are inferred by using multiple sensors input, e.g. multiple sunblinds implies multiple sunblinds states that will be summed. But we will not extend the description of such details. The lightning state of the room itself may be described as ("dark" or "natural lightned" or "artificially lightned").

dark if lightning of the room = "dark"

natural lightned if lightning of the room = "lighted" $\wedge$ outside lightning = "lighted" $\wedge$ sunblinds = "open".

artificially lightned if lightning of the room = "lighted" $\wedge$ outside lightning = "dark".

The occupation of the room
In order to determine if the actual occupation of the room (**"empty" or "occupied"**) we need to merge the inputs of the presence sensors of the room. Each presence sensor can determine if something moves around it.

```
POST central_unit
  <MESSAGE>
    <TYPE>event</TYPE>
    <FROM>sensor_id</FROM>
    <CONTENT>presence:('true'|'false')</CONTENT/>
  </MESSAGE>
```

Those sensors inputs are merge to produce a single state of the room. If S is the set of sensors used then the state is infered by the following equations.

occupied if $\bigvee_{s \in S} s_{state} = True$

empty if $\bigvee_{s \in S} s_{state} = False$

The actual weather state inside and outside
(temperature, lightning and precipitations). This context depends on many different sensors input and merging. The first subcontext to determine how the temperature state outside and for each room is as (**"cold", "comfortable" or "hot"**). Those terms are based on a user-determined threshold. Each subcontext can be infered by using data provided on the Internet or by using outside temperature sensors. If S is the set of sensors used outside or inside then the temperature is determined by the following equation.s

hot|comfort|cold if $\frac{\sum_{s \in S} s_{value}}{|S|} \in [min_{value}, max_{value}]$

The precipitations state (**"rainy" or "dry"**) be infered by using values retrived over the Internet.

Which is the part of the day ?
The actual date and time is enough to infer such an information. The part of the day can be **"morning", "noon", "afternoon", "evening" or "night"**

Is a car waiting at the garage doors ?
If an authorized car approches the RFID reader then the door can be opened. The message send by the RFID reader have the following form.

```
POST central_unit
<MESSAGE>
  <TYPE>event</TYPE>
  <FROM>RFID_reader</FROM>
  <CONTENT>
```

```
    presence:('true'|'false'),
    car:('allowed','forbidden')
  </CONTENT/>
</MESSAGE>
```

If a car with a RFID tag approaches the RFID reader then the presence field turn to "true". If the tag is authorized then the car field turn to "allowed".

What is the actual consumption of the whole building ?

This context might be inferred by using the three following sensors, the energy consumption sensor which measure the electric consumption. And the water consumption and temperature sensors that measure the current state of the water status of the house. Those sensors can be used to infer two subcontexts that are the current status and the overall status. The current status display the consumption of the building or the apartment in real time. The second status can be computed by aggregating the sensors value and store them into a small database. The user can then access those values over the web interface provided by the central unit.

We choose those six contexts because they cover many facets of the central unit (web interface, data aggregation or even WEB information retrival).

4.3 Actuators

Admit now that each context has been computer by using the sensor data. We must now describe how an actual context combined with new sensors data coming in will react. This can be done via listening all the possibilities and the messages sent by the central unit to the actuators. Here we only gives a few examples for the messages as their are very similar.

Do activate lightning **if** somebody enters the room $\wedge$ the room is dark.

```
POST light_switch_id
<MESSAGE>
  <TYPE>do</TYPE>
  <FROM>central_unit</FROM>
  <CONTENT>light:'on'</CONTENT/>
</MESSAGE>
```

Do deactivate lightning **if** the room becomes empty $\wedge$ the room is lightened.

Do reduce heating **if** room is usually empty for a quite long period $\wedge$ nobody is inside.

```
POST thermostat
<MESSAGE>
  <TYPE>do</TYPE>
  <FROM>central_unit</FROM>
  <CONTENT>set_temp:'value'</CONTENT/>
</MESSAGE>
```

Do reduce heating **if** outside temperature $>$ comfort temperature $\pm$ threshold

Do augment heating **if** temperature $<$ comfort temperature $\pm$ threshold.

Do open doors **if** car is at the parking doors $\wedge$ door closed.

Do close the doors **if** timer for doors is over $\wedge$ no car at the parking doors.

```
POST door_actuator
  <MESSAGE>
    <TYPE>do</TYPE>
    <FROM>central_unit</FROM>
    <CONTENT>door[0]:'open'</CONTENT/>
  </MESSAGE>
```

Do close sunblinds **if** lightning outside $<$ threshold $\wedge$ part of the day = 'evening $\vee$ night'.

Do open sunblinds **if** sunblinds closed $\wedge$ part of the day != 'evening $\vee$ night' $\wedge$ someone enters the room.

Some cases need to be specified to understand how the actuators actually works because they might lead to conflicts. This is specially the case for lighting actuators. We are going to specify here two types of lightning actuators and the doors actuators. The first type of lightning switch is called "schema 0" and it the most simple case of switch possible. Its structure is shown by figure 2. In that case it is possible to let the actual wire structe as it is and only remplace the switch by our smart switch.

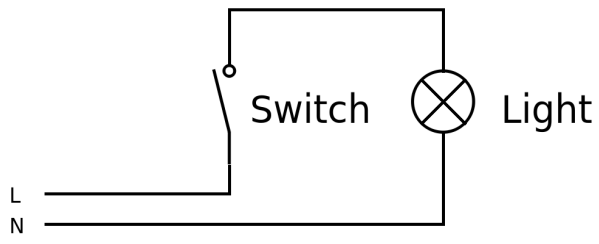

Figure 2: Schema 0

A problem occurs when dealing with more complex wired structures. We here only discuss of the "shema 3" type that allows to control one single light with two switches. Its structure is shown by figure 3. Here the problem is that the structure is now composed of multiple wire. But we can trick this situation by allowing each light actuator unit to send a message to the other units that compose also the same lightning structure. Each time a switch is ordered to be turned "on" or "off" it relays the same message to all the others concerned light actuators. We can then remove wires and keep only one direct link between all switches as shown by figure 4

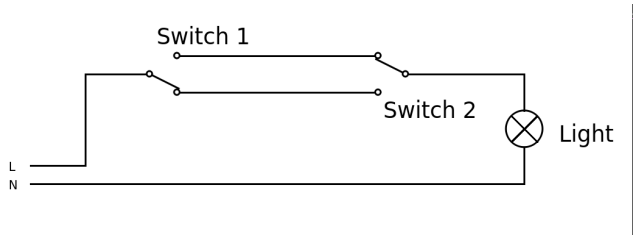

Figure 3: Schema 3

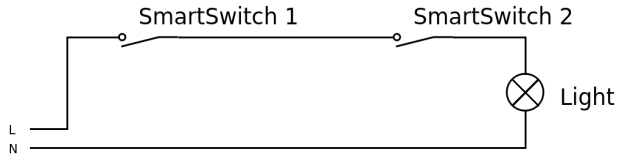

Figure 4: Smart schema 3

4.4 Web-based manual controller

We think that our system must have an interface in order to offer a manual management of it. As the transmission of the data sensor is based on the standard HTTP protocol we can deduce that the network also support web-based application. Providing a web-based interface can be useful because with nowadays technologies we can provide with adapting template the same website for mobile and desktop devices. This precision is important because smartphones are widely used they can be used to access the central unit more easily. Furthermore using a web-based application with a good authentication protocol, such as HTTPS, can make the interface available over the Internet. The users can thus control their homes at work or switching off the lights forgotten in the morning.

5. CONCLUSION

6. ACKNOWLEDGMENTS