Energy Smart Office

Rik Schaaf & Maarten Kollenstart

July 9, 2015

Abstract

This research proposes a system to combine data from multiple kind of sensors to control office appliances to reduce energy consumption. By combining the data of multiple sensors the system is able to take decisions more accurately. Timeouts can be reduced compared to single sensor systems. The system is able to control many different appliances, as long as they can be controlled via an Representational state transfer (REST) interface.

The system can be effective not only for reducing energy consumption, but also for more general home automation and medical applications.

Contents

1 Introduction **4**

2 Research question **5**

3 Analysis **6**

3.1 Accuracy enhancement 6

3.2 Ease of use 7

3.3 Other research 7

3.4 Other projects 8

4 Methods and design **10**

4.1 Data flow and data manipulation 10

4.2 Controller 12

4.2.1 Registration 13

4.2.2 Adapter 13

4.2.3 Checker group 14

4.3 Configuration 15

4.3.1 Example 16

5 Implementation **18**

5.1 Data consumers 18

5.2 Queue 18

5.3 Database 19

5.4 Controller class 19

5.5 Registration Class 20

5.6 ThresholdChecker classes 20

5.7 Adapter classes 21
5.8 JSON 21
5.9 YAML 22
5.9.1 The config file 22
5.9.2 The parser 24

6 Tests **25**
6.1 Test environment 25
6.2 Energy savings 26
6.3 Accuracy 27

7 Conclusion and discussion **28**

8 Future work **30**
8.1 Machine learning 30
8.2 Support for different database systems 30
8.3 More general adapters 30
8.4 More than only energy saving 30
8.5 User Interface 31

1 Introduction

Nowadays, the reduction of energy consumption plays a great role in our lives. We are looking for alternative energy sources, because of our ever increasing population and consumption per person. We also try to find new solutions, because it is estimated that we are running out of fossil fuels faster and faster.

On the other hand we try to reduce or at least halt the growth of the amount of energy we need in our daily lives. Energy consumption in industry, commerce, traffic and at home take up most of the total energy consumption in this world[13]. This research tries to find a ways of reducing the energy consumption in one particular part of the commercial sector.

Attempt have been made in the past to reduce the energy consumption in this sector. Automatic lighting for instance prevents lights from being on when there is no one around, more efficient heating systems have been created, computers have been introduced to the sleep mode for either the monitors or the computer itself, and lots of other systems were made to reduce energy.

With all these existing technologies, it is a shame that they don't harmoniously work together. A lamp is usually only controlled by a presence sensor (for instance a passive infrared-sensor) and occasionally a luminance sensor, but don't make use of other sensors in the room. Examples could be computer activity sensors and location sensors in smartphones.

Combining data from multiple sources has the potential to increase the accuracy of the system as a whole. It could for instance be possible to combine Bluetooth beacons location technology[3] with a computers activity to determine more accurately whether a person is actively using a computer.

Increasing demands of users and companies to control appliances in their homes or in offices from their smartphones lead to improvement of wireless communication standards and the costs of creating a wireless sensor network, for example with Zigbee or 433MHz communication, are still decreasing. This also involves so called smart appliances, like smart refrigerators and smart washing machines, that can be monitored and controlled from the internet with mobile apps.

2 Research question

RQ1: Can energy consumption be reduced in an office environment by combining data of multiple sensors?

RQ2: Is the usage of multiple sensors increasing the accuracy of decisions regarding energy saving in an office?

The main goal in this research is to reduce the energy consumption in an office environment through integrating multiple sensors that are frequently present in such an office. As the question that this research tries to answer is if such a system is more efficient in the reduction of energy use and whether the system is conceived as more favorable by the user.

Part of this question is whether or not combining data from more sensors could improve the accuracy of the decision if appliances should be turned on or off. The meaning of accuracy in this research is the fact to reduce errors made by single sensor solutions. This includes the false negatives, the moments that appliances could have been turned off but weren't, and also false positives, the moments that users were expecting the appliance to be on but were turned off by the system. The interesting part is finding a balance between energy saving and user friendliness.

A requirement of the system is that it should be easy to install and configure. It should be easy to add or remove sensors and appliances from the system.

3 Analysis

It has been estimated that 8 to 50 percent of energy usage is consumed in the commercial sector, most of which is due to lighting, electrical equipment, heating/cooling, and elevator usage. [16]. In this research the focus will be on the first two, since their usage can be easily monitored.

3.1 Accuracy enhancement

Most of the other research, discussed below, used a sensor network with a specific kind of sensors. For example passive infrared sensors to detect whether someone is in the office at that moment or luminance sensor to detect the illumination in the office to determine if there is enough light in the office without the lights turned on.

There is no well-known research that tries to combine the data of a network of diverse sensors to get a better image whether or not appliances could be turned off or not. However, combining this data could lead to a better decision algorithm that takes into account a multitude of data points to decide to turn appliances off.

Due to the fact that there are a large amount of situations people can be in when they're in an office single sensors are not able to give the right decision in all of those situations, especially when situations occur that are not taken into account when the thresholds are set for such sensor. Using a grid of sensors could then give a better understanding of the situation then a single sensor with two possible outcomes: true and false. With this larger data set belonging to the situation the decision whether or not appliances are turned on or off can be made with a greater confidence and thus the thresholds can be adapted to this confidence. When the thresholds can be adapted the decision to turn an appliance off can be made earlier and thus more energy could be saved.

The range of sensors that could be used in such a system is very wide. Classic sensors can be used, such as light sensors, presence sensors and temperature sensors, to get a general impression of an office. However, more complex sensors could also be used to get values of more specific elements in the office, for example: using a phones location sensors to detect which person is present in the office or using smart thermostats like the Nest from Google[6].

3.2 Ease of use

As a requirement it should be easy to add new sensors to the system. This includes the ability to use different kinds of sensors and be able to set their data threshold. These sensors can rage from detecting people and/or their activity to temperature, lighting, power consumption or weather sensors.

Customizability is needed to be able to use the different kinds of data that the sensors use. Some use numbers to describe the sensor state. LUX-sensors for instance give the lux value it detects. Other sensors use a string as state. Lazy Sleep[17]/Sleepy for Linux[8] uses as activity sensor the values "idle", "active" and "inactive".

3.3 Other research

The research of Michele Magno et al.[14] is partly similar to this research. The paper is stating that the amount of homes and offices that use LED lighting solutions is relatively low because the initial purchase is quite pricy and combining that with a system that intelligently turns those lighting solutions on and off are not widely available. The paper presents the design and implementation of a system of motion sensors and light sensors in a low-power wireless solution using Zigbee communication, which is an open standard for wireless sensor network communication. The paper also holds into account the ease of use, that the system could be implemented in an office quite easily without having deep understanding of the system. The paper claims a reduce of total power consumption of 55% during a six month period starting from the winter end ending in the summer, in the spring months the reduce in power consumption raise as high as 69%. The drawback of the paper is that they went from a traditional lighting solution to an LED lighting solution which in itself saves a lot of energy. Also, only motion sensors and light sensors are used to control the lighting and no other information is used in the proposed system. That's where the focus of this article is, combining more data from a multitude of sensors to get a better picture of the situation. Besides that this article also tries to create a solution that could be used for more than only controlling lighting.

The paper of Moonok Choi et al.[3] presents a way to use Bluetooth Beacons to determine whether users are present at the office or not. The Bluetooth Beacons are fixed Bluetooth devices with a known location placed on strategic points in an office. With this information PC's, monitors and lights in an office are controlled. The location done by a mobile app that detects which Bluetooth Beacon is the nearest and the app knows the beacon nearest to the seat of the employer and tells the system to turn on or off the users PC and monitor accordingly. The benefit of this approach is that the system also knows which employees are present at the office, but if a employee temporarily wants to work

on a coworkers PC that could give problems if that coworker leaves the office and the system will turn off that PC. The proposed system works only with the Bluetooth Beacons and only with turning off PC's, monitors and lighting and no other appliances.

A more Artificial Intelligence based solution is proposed by Georgievski et al.[5] which combines an ontology-based technique to recognize activity in an office and Artificial Planning to determine whether appliances should be turned on or off. The solution is capable of handling multiple-user and multiple-area situations in detecting activity of users.

Another quite simple approach is the one proposed by Kaneko et al.[12] that uses an access control system to check whether or not someone is present at that floor of the building. This is a very easy to implement system and could be implemented in a lot offices, as many offices already use a access control system to grant access to offices to employees. Due to the nature of this simplicity the energy saving is not as high as other research because all the lights on a floor are on when only one employee is present at that floor. However, this system could work well when for example a heating, ventilation, and air conditioning (HVAC) system is controlled by the solution and the HVAC system only works on a floor based level.

An research not focused on presence or activity is done by Ahmadi et al.[1], this research is about controlling HVAC systems. In the research a sensor network with temperature, humidity and CO_2 registrations is proposed, using the same TelosB platform for the sensors as in this paper. The research mainly focuses on gathering the data instead of using the data for lowering the energy consumption.

3.4 Other projects

There are already some consumer electronic devices on the market that are trying to save energy in home environments. For example the Nest Learning Thermostat from Google Inc.[6] which is a programmable thermostat that learns from user behavior what heating level the thermostat should be on. This system can also be controlled from a smartphone where the user can turn the heating already on just before he arrives at home. And see statistics of the energy usage of the house so that the user can identify which appliances are using a lot of energy. This system is for now a relative small solution with only the thermostat. Besides the Nest Learning Thermostat there are a few other smart thermostats like the Honeywell Lyric[10], the EcoBee3 Smart Thermostat[4], these thermostats can be controlled with mobile apps and most of them use for example the location of the device to turn on or off the heating. All of these thermostats are connected to the Internet to get latest wheather information and compares data between customers to give customers insight in their potentially

waste of energy.

Another solution is the initiative started on Kickstarter named Homey[9]. Homey is a smart home application that can control a large variety of products like Lights, Multimedia, Appliances, Security and the smart thermostats discussed before. It supports almost any communication technique for home automation, including WiFi, Bluetooth, Zigbee, Infrared and 433MHz. This makes it an interesting solution to control a complete house or office.

All of these projects and products are mainly focused on individuals and personal use and not on saving energy in an office environment, but the techniques used could also improve office related projects and research.

4 Methods and design

Now the design of the proposed system is discussed to show the idea of the system without stepping too much into implementation specifics and keep the design as general as possible.

4.1 Data flow and data manipulation

The system is build up around a couple of separate blocks, the general layout of the system is shown in figure 1 on the following page. In this diagram there are 6 main components shown: the sensors, the data consumers, the queue, the database, the controller, and the appliances. This diagram also shows the dataflow between the components.

Our design makes use of a currently already available database with a queue to add data and a REST interface to get the data from the database.

The data of the sensors is retrieved by data consumers that are kept as simple as possible, the only operation the data consumer does is retrieve the data and send it through to the queue. However, not all data is presented the same way by the sensors so the data consumers have to convert the data to a standardised presentation form. The presentation form is the JavaScript Object Notation (JSON)[11] where one sensordata object contains the following name/value pairs:

timestamp The timestamp on which the sensordata value was generated, represented as UNIX timestamp

process_id A UUID of the process when the sensordata value was generated, this value is not used in the current system.

sensor_id The identifier for the sensortype, presented in a UUID. Sensors that are of the same sensortype share the same sensor identifier.

instance_id The identifier for the specific sensor, presented in a UUID. All sensors must have a unique instance identifier. Sensors of the same type don't share instance identifier.

value A base64 encoded value of the sensordata value. Therefore, this name/-value pair is always of the type String, even if the sensordata value is in fact an integer.

For identifiers the universally unique identifier (UUID[18]) standard is used, to ensure all identifiers are unique. Listing 1 on page 12 shows an example of

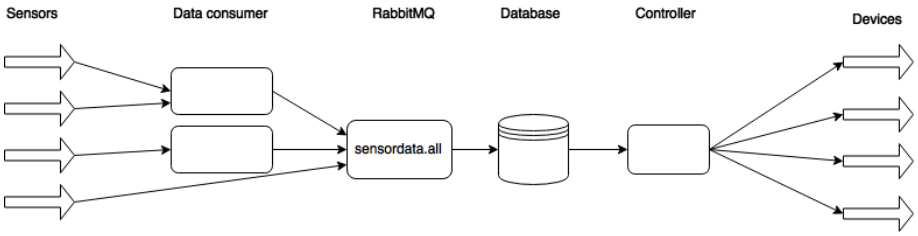

Figure 1: A diagram of the whole system

such a sensordata package with PIR-sensor data that is sent to the queue. The current queue setup requires the values from the JSON to be base64 encoded[2].

```
{
"timestamp": 1434646643,
"process_id": "35aa451f-b738-420f-8ec6-a3534b68b0df",
"sensor_id": "aedab570-f8b7-11e4-b939-0800200c9a66",
"instance_id": "e4ed25b1-f959-11e4-b939-0800200c9a66",
"value": "MTU="
}
```

Listing 1: Sample JSON package

The queue is used to distribute the sensordata packages evenly to the database. The queue system is a system with a lot of producers, the data consumers that consumed data from the sensors and publish this data to the queue, and a single consumer, the database storing the sensordata permanently. The queue could in fact be consumed with multiple threads, but for the rest of this paper it is assumed that the database is a single component. Therefore the database as a whole is considered as the only consumer of the queue.

The controller receives it's data from the database. This is done via a REST communication interface provided with the database with a single endpoint that is used to retrieve the latest value of one specific sensor. This endpoint requires two parameters to be able to identify the sensor which data is requested, namely the previously mentioned sensor identifier and instance identifier. The database provides the requested data in a similar form as the JSON package described in listing 1 on page 12. However, there are two pairs different from that example: the timestamp is not delivered as UNIX timestamp but as string in the form "YY/mm/dd-H:i:s", the other difference is that the values are not longer base64 encoded but are represented as normal strings. These differences were already present in the test environment.

The controller then analyses the data and makes the decision whether or not devices should be activated or deactivated. The analyses is discussed later, the activation or deactivation is done via POST messages via the REST interface of the device or via the REST interface of the control unit of the device.

4.2 Controller

The controller is the main part of the software in this paper. The controller is responsible for analysing sensordata to be able to make the correct decision whether or not to activate appliances. The used design was inspired by the model-view-controller (MVC) pattern. Some parts of the controller require a bit more explanation, these parts are: the Registrations, the Adapters, and Checker groups.

4.2.1 Registration

Registrations are objects that are responsible for taking care of a subset of sensors and appliances. These can be considered the controller part of the MVC pattern. The registration objects are observers of the controller, the controller notifies all registration objects when it successfully updated the latest sensor values in a map that is accessible for all registration objects. After being notified by the controller, the registration objects loop through all checker groups and let them decide whether or not the adapter belonging to the registration should be activated or deactivated.

4.2.2 Adapter

Adapters are responsible for making a uniform activating and deactivating platform so that the registration objects are able to activate and deactivate appliances without knowing how such appliance should be activated or deactivated. Adapters will in most cases use some sort of web interface, like REST or simple UDP/TCP packets, to communicate with the appliances. The adapters can be considered the 'view' part of the MVC pattern.

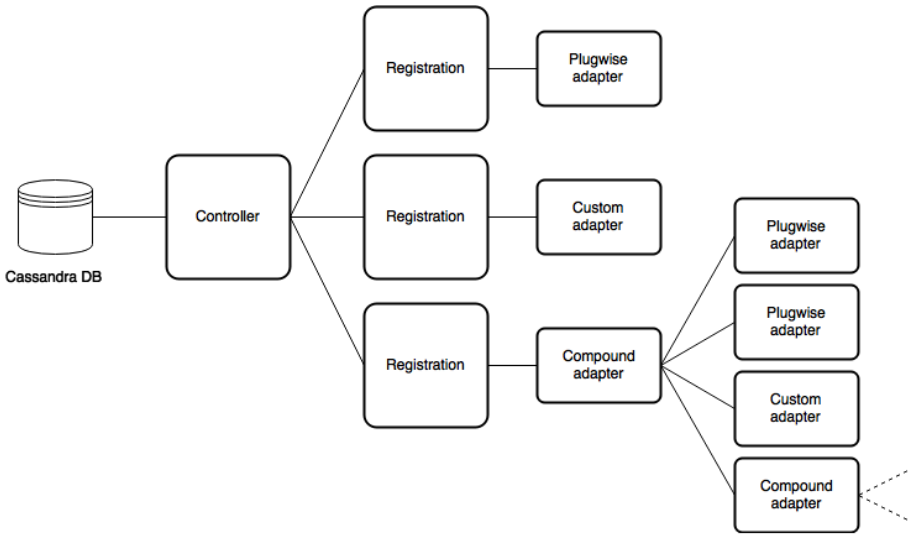

Figure 2: A diagram of the adapters in registrations

An other type of adapter is the compound adapter. It can contain multiple adapters that can be activated or deactivated at the same time. Since a compound adapter contains multiple adapter there should be a possibility to negate

one of the adapters, for example to turn off the main lights and turn on emergency lights. This is done via a negation adapter which contains a so called child adapter, the negation adapter switches the activate and deactivate behaviour.

4.2.3 Checker group

Checker groups are the main part of the decision taking within a registration object. These can be considered the model part in the MVC pattern. In one registration object there are multiple checker groups, each checker group consists of one or more checkers that give a Boolean whether or not a threshold is met for a certain sensor. If all checkers in a checker group return **true** that checker group will lead to activation of the adapter belonging to the registration object. If all checker groups have at least one checker that returns **false**, then the adapter belonging to the registration will be deactivated.

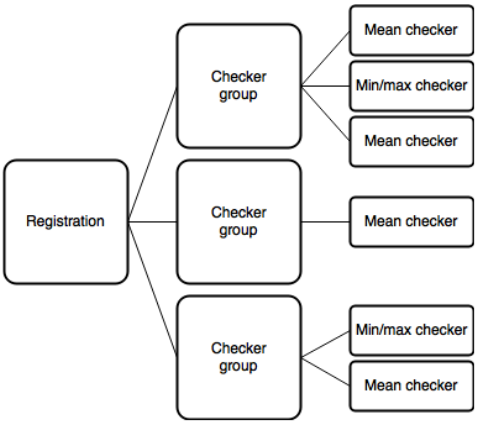

Figure 3: A diagram of the checker groups in registratons

There are two types of checkers: mean checkers and min/max checkers. Lets assume that n values of a sensor will be determined by both checkers. The mean checkers will average the n values and check if that mean value is higher than the threshold set for that checker. That way there can be checked if over time a certain threshold is met, which could be useful for luminance sensors. For instance, when someone passes by a luminance sensor the reading of that sensor is most likely to be a bit lower due to shadows, taking an average of for example 15 minutes can filter out such small fluctuations in the sensor values.

Min/max checkers are checkers that take the minimum or maximum value of n sensor values. This min or max value is then checked against a threshold to determine the result of the checker. Such min/max checkers can be used to see, for instance with a PIR sensor, whether or not there was activity measured

in the past n values. The default behavior is that the checker returns true if there exist at least one value that exceeds the threshold which is a existential quantification. Negating that behavior leads to the universal quantification which means that there are no values above the threshold.

Both mean checkers and min/max checkers have a switch to return true when the values are above a certain threshold or to return true when values are below a certain threshold. This way a wide variety of checkers could be constructed.

Each checker is configured with just one sensor identifier and one instance identifier, because thresholds are bound to single sensors. However, multiple checkers for the same sensor can be configured in a checker group.

4.3 Configuration

Installing and configuring the system is supposed to be quite simple. That's why a single configuration file is designed for the system. Given the previously described structure of the system the following structure for the configuration file can be defined.

It starts with a list of registrations. In each registration there is exactly one adapter. Of course this adapter could be a compound adapter and more adapters could be added to this compound adapter. The information for the adapter should be sufficient to be able to activate and deactivate the adapter. For each kind of adapters these settings will be different.

Then each registration needs a list of checker groups, which could be given a name to identify more easily which group leads to activate an adapter. The other setting for a checker group is a list of the actual checkers in that group.

A checker always consists of the following properties: sensor identifier, instance identifier, threshold, and a switch to identify whether values should be higher than the threshold or not. Then specific for a min/max checker a data count is needed to specify the amount of values that are checked and a switch to be able to negate the result of the checker. For a mean checker the only extra property is the data count, a negation isn't necessary here because that can be done by the general property of the checker.

This are the only settings needed for the system to run correctly. In the implementation of this project there will be created a Plugwise adapter and a Custom adapter. The Plugwise adapter is specific for the Plugwise devices, the custom adapter can be configured with REST endpoints for activating and deactivating and a sensor identifier and instance identifier to retrieve the state of the adapter.

4.3.1 Example

In figure 4 a possible configuration is shown. It is an example of a registration, that consists of a compound adapter and two checker groups.

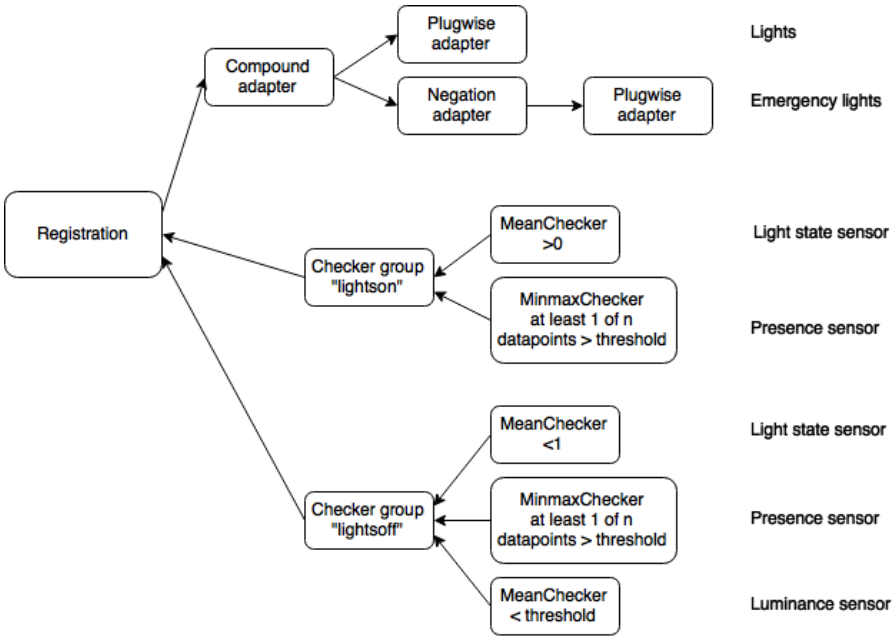

Figure 4: An example of a configuration that could be used in the system

The compound adapter contains two adapters. The first is a plugwise adapter that controls the main lighting. The second is a negation adapter that contains the plugwise adapter of the emergency lighting. This way, when the main lighting is turned off, the emergency lighting turns on and vice versa.

The two checker groups are named “lightson” and “lightsoff”. The lightson group has 2 conditions. One of them is that the lights are on. This is checked with a mean checker that checks if the latest state sensor data point is higher than 0 (or in other words: the state is 1). The other condition is that at least one of the n latest data points of the presence sensor contains a value higher than the threshold.

The lightsoff group has three checkers. The first, a mean checker, checks if the latest state data point in lower than 1 (so the state is 0). The second performs the same presence sensor check as previously described. The third checks if the latest n data points from the luminance sensor have an average that is lower

than the threshold, which prevents the lights from going on in a room that is sufficiently lit by natural light.

5 Implementation

The implementation of the software is done in the object oriented language Java. The project is set up with a maven project to make dependencies load easily and development can be done on multiple systems without needing to download dependencies manually.

In this section the main parts of the system will be discussed to give a better insight of the actual implementation of the system used to test the design. The sections are ordered in the way the data flows through the system.

5.1 Data consumers

The data consumers are the first part of the software. As discussed earlier these consumers are responsible for streamlining the data flow between sensors and the database.

The consumer created for the tests is a consumer that consumes data of the LUX and PIR sensor. The values of the sensors are consumed from a RabbitMQ[15] queue. The data received from the RabbitMQ is a JSON string identical to the one mentioned in listing 1 on page 12, this string is parsed using the Gson Java Library from Google[7]. Gson is used over just using the Java JSON API to make serialization and deserialization easier. To use the Gson library a container class, named RMQSensorData, has to be made containing fields identical to the name/value pairs of the JSON object. Because JSON doesn't support UUID datatypes and doesn't automatically converts base64 encoded strings to normal string, a Gson data serializer and deserializer must be created to convert the JSON string to a RMQSensorData object.

The data consumer is also capable of changing the frequency of messages that are published to the RabbitMQ queue so that all data in the database has the same frequency, except from binary data like on and off values of Plugwise devices.

5.2 Queue

A RabbitMQ queue is used to implement the queue, because it's easy to use and in the test environment was already present. The sensordata.topic exchange is the exchange where messages should be published in order to be consumed later by the Cassandra Consumer, which is discussed in the next section.

The queue is in fact a load balancer for the database, when there are a lot of messages within a small window the RabbitMQ server can hold the messages

temporarily until the database consumer consumes these messages.

5.3 Database

The database is implemented with Apache Cassandra, which is able to handle large amounts of data. That point is very useful for sensordata because the data of sensors is data that is generated at a fixed interval and runs continuously that could grow to a lot of entries in the database when all data is saved.

The Cassandra database is already set up in the test environment. In this test environment there is also a REST interface available that can be used to retrieve data from the database. The available endpoints in the system are: between, latest, and before. The only endpoint used in this implementation is the latest endpoint that needs a sensor identifier and an instance identifier to identify the requested sensor. The result of the latest endpoint is a JSON string similar to listing 1 on page 12. Because the JSON message slightly differs from the message that is consumed from the RabbitMQ queue a new SensorData class and serializer and deserializer are created that act similar to the ones created in the data consumer.

5.4 Controller class

The Controller is the class that is responsible for retrieving data of sensors and notifying all registrations that there is new data they can analyse. Therefore, the Controller class is obviously an subclass of the default java util class Observable.

The class runs as a thread because it isn't retrieving data constantly but at a specific frequency, so a infinite while loop is used with a sleep at the end of each loop. This makes sure the program only retrieves new data from the Cassandra REST interface, which means a reduced network usage of the program.

The Controller class holds a set of sensor identifier and instance identifier called the sensorKeys, this set represents all the sensors that are currently used by any checker groups. This set is updated by the checker groups itself, via the getSensorKeys method that returns the sensorKeys object. The set is used by the controller to know which sensors values are supposed to be retrieved.

The retrieved data is set in a map, called the sensorDataMap, with keys identical to the sensorKeys set and the values are the SensorData objects that are retrieved from the Cassandra REST interface. After storing all data in the sensorDataMap all the observers that observe the Controller are notified that new information is available for them.

5.5 Registration Class

The observers of the Controller are the Registrations. This Registration abstract class holds one Adapter class. At the moment it has two implementing classes: SimpleRegistration and ComplexRegistration. The latter contains a map of named CheckerGroup objects. These CheckerGroup objects are a simple wrapper for a HashMap of Sensors and ThresholdCheckers.

Because the Registration objects are observers of the Controller, the Registration class has an update method that is fired after the controller notified it's observers. In this update method the Registration class makes a decision to activate or deactivate the adapter.

```
function update(controller)
    gather the new latest values for the checkers from the controller
    forall checkergroups do
        forall checker in checkergroup do
            if checker returns false then
                deactivate the adapter
                return
        end
    end
    activate the adapter
end function
```

Listing 2: Pseudo code registration update algorithm

This is done in the following way. First there will be looped over all CheckerGroup objects that are present in the map mentioned before. Then all ThresholdCheckers in the CheckerGroup will be looped over, if one of them returns false that inner loop will be stopped and the program continues with the next CheckerGroup. If none of the ThresholdCheckers in a group returned false then the activate method of the adapter will be called and the method will return. If none of the CheckerGroups lead to activate the adapter the Registration will call the deactivate method of the adapter.

5.6 ThresholdChecker classes

The ThesholdChecker abstract class is the class that represents objects that check values for a single sensor whether or not a certain threshold is met. Because thresholds for sensors can span multiple data points there is a linked list inside the Thresholdchecker class with SensorData objects, the reason this is a linked list is because this way the SensorData the order of SensorData is preserved, which is useful to contain a list of n most recent SensorData objects of a certain sensor.

For subclasses that implement ThresholdChecker only the method check has to

be implemented, this in fact is the logic of the threshold checker. Currently there are two thresholdcheckers implemented, namely the MeanThresholdChecker and the MinMaxThresholdChecker.

The MeanThresholdChecker only contains a constructor and the implementation of the method check, where inside the method check the mean value of all the values of SensorData objects inside the linked list are averaged, then a simple check against the sensor threshold is made to return true or false.

The MinMaxThresholdChecker just keeps either the minimum value or the maximum value of the SensorData values from the linked list. This minimum or maximum is checked against the sensor threshold.

5.7 Adapter classes

Adapters are the classes that differ the most between each other. Because for each device there can be a different adapter. However, Adapters should follow a certain standard and therefore a interface for Adapters is created.

Three methods are required for each Adapter and those methods are used by the Registration to communicate with the adapters. These methods are activate, deactivate and getState. The activate and deactivate methods are quite self explanatory, these methods are supposed to actually activate or deactivate devices and should return the result of this action. The getState method should return an integer that represents the state of the device, this is used to check whether or not a device state is actually switched or not.

5.8 JSON

The conversion of JSON string into Objects is done with the Google Gson Library. However, to be able to convert the strings directly to SensorData objects there had to be created a serializer and deserializer. These serializers are made relatively easily, listed below is the deserialization method that converts a JsonElement je into an SensorData object:

```
public SensorData deserialize(JsonElement je, Type type,
    JsonDeserializationContext jdc) throws JsonParseException {
  try {
    JsonObject jo = je.getAsJsonObject();
    SimpleDateFormat df = new SimpleDateFormat("dd/MM/yyyy-HH:mm:ss
        ");
    Date timestamp = df.parse(jo.get("timestamp").getAsString());
    UUID processId = UUID.fromString(jo.get("process_id").
        getAsString());
    UUID sensorId = UUID.fromString(jo.get("sensor_id").getAsString
        ());
```

```
    UUID instanceId = UUID.fromString(jo.get("instance_id").
        getAsString());
    String value = jo.get("value").getAsString();
    return new SensorData(timestamp, processId, sensorId,
        instanceId, value);
  } catch (ParseException ex) {
    Logger.getLogger(SensorDataDeserializer.class.getName()).log(
        Level.SEVERE, null, ex);
  }
  return null;
}
```

Listing 3: SensorData deserializer

As you can see the only thing that needs to be done is parse the JSON primitive types into the types needed for the SensorData object, which are the UUIDs that are created from strings and the Date that is parsed with an SimpleDateFormat object.

With this serializer and deserializer it is now possible to create a SensorData object from a JSON string with the following lines of code:

```
Gson gson = new GsonBuilder().registerTypeAdapter(SensorData.class,
    new SensorDataDeserializer()).create();
SensorData sd = gson.fromJson(result, SensorData.class);
```

Listing 4: String to SensorData conversion

The first is to register the deserialization object to the Gson Builder and the second is the method that converts the string to the object.

5.9 YAML

For the configuration of the sensors the program uses a configuration file. This file uses the YAML language[19], which is an relatively easy to read data serialization format. Because at this point there isn't a graphical user interface, a YAML file makes it easy for users to change settings. Alternatively XML or JSON could have been used, but these are harder to read by humans. To read from this YAML file, a library named SnakeYAML was used, together with a self written YAML file, node and nodelist java class for easy file traversing through the file.

5.9.1 The config file

The config file has three base nodes: updatetimeout, cassandrahost and registrations. The update timeout node specifies how often the controller has to

fetch data from the cassandra database. The cassandrahost node contains the ip address (and if needed the portnumber) of the REST interface to the cassandra database.

Adapters
The registrations node contains a list of registrations. All registrations have an adapter node that can be of the type plugwise, compound, negation and custom. Plugwise adapters have a host node for activation and deactivation, a mac node to specify the mac address of the device, a stateSensorId node and a stateInstanceId node for getting the state of the device.

Apart from its type, the compound adapter only contains a adapters node with a list of other adapters. These can be of any of the previously mentioned types. The negation adapter contains an adapter node that specifies which adapter needs to be negated. This can also be an adapter of all other adapter types. Negating a negation adapter does work, but double negation is probably something the user doesn't want to do. The user is free to make the adapter tree as large as he would like.

The last adapter, the custom adapter, contains three nodes: activate, deactivate and state. All three contain a host, path, endpoint and param node to specify to which REST interface it wants to talk to activate, deactivate or request the state of the device. The state node also contains a responsetype node which should be either set to raw or sensordata. Raw specifies that the returned data only contains the state value itself. Sensordata means that the returned state data is wrapped in a SensorData JSON object, as described before. Due to the fact that our system works with numeric threshold values and that cassandra database can also return non-numeric string values, there is also a customvalueformat node. This contains a list of stringvalue/numbervalue pairs that makes the program read the string value as its numeric counterpart (for instance "on" and "off" or "active" and "inactive" to 1 and 0). If the response only contains numeric values, the customvalueformat node list can be left empty.

Registration type
Registrations can be of either of the following types: simple or complex. A simple registration is basicly a restricted, but easier to use form of the complex registration. It has the same fuctionality as a complex registration with only one checkergroup that contains checkers with only the latest data point from the database. Therefore there is no difference between a mean- or a minmax-checker, due to there only being one data point.

In the YAML file the registration list item of the simple type has a sensors node that contains a list of all sensors that should be checked for the adapter to activate. These sensors have a sensorId and instanceId node to get the data, a

threshold node to set the threshold for activation and a greater boolean node to specify if the data value should be greater or smaller than the threshold value.

For registration list items of the complex type there is a groups node that contains all checker groups. Every group list item has a name node and a sensorcheckers node with a list of sensors. Just like in a simple registration, the sensors contain a sensorId, instanceId, threshold and greater node. If greater is true this specifies the minimum (everything has to be greater than the minimum) and if false it defines the maximum (everything has to be smaller than the maximum). Unlike the simple registration it also contains a customvalueformat node that works just like previously described in the custom adapter. It also contains a checker node with therein a type node that specifies if it is a mean or a minmax checker. The checker node also contains a datacount node that specifies how many datapoints are used in the checker. On top of that, the minmax checker also has a negation node to handle both existential and universal relations (all beneath the maximum/above the minimum or at least one above the maximum/beneath the minimum).

5.9.2 The parser

The YAML file is parsed in the class ConfigParser. This class has a static method parseConfig that takes the controller's sensorKeys field and returns a list of registrations. During its execution it sets the timout for the controller, sets the host of the REST interface to the cassandra database, creates all the registrations that are specified in the YAML file and puts all needed sensorId/instanceId pairs into the provided sensorKeys variable, so that the controller knows what data to gather from the database.

6 Tests

6.1 Test environment

For testing the software an office was used with a 2x2 TL lighting solution and a desktop computer with two monitors. The lighting solution uses around 150 Watts when turned on. The desktop computer uses on average 50 Watts when it's used and 10 Watts when it's in sleep mode. The monitors use around 25 Watt each and use between 2 and 3 Watts when the backlight is turned off. These energy consumption values are all measured via Plugwise devices that measure the energy consumption at the electric outlet. In the test environment all of the TL lights were connected to a Plugwise adapter which enabled the system to turn on and off the lights via a REST interface, the computer is turned to sleep and turned back on via Sleepy for Linux.

The sensors that are available are: one presence (PIR) sensor, one luminance (LUX) sensor, and Sleepy for Linux activity sensor. An PIR motion detection sensor detects changes in the amount of infrared radiation reaching the sensor. The luminance sensor detects the amount of light falling on the sensor, by putting this sensor in a place without direct sunlight the overall lightlevel of the office can be determined. The Sleepy for Linux activity sensor detects activity on a pc by detecting any user interaction with the pc.

Figure 5: The test environment

Because the windows of the office face north, there isn't any direct sunlight in the office and almost always the lights should be on when someone is present in the office. And the TL lights can't be dimmed to get a steady luminance level.

The office that was used for testing isn't being used full time so there couldn't be tests performed on full workdays for a longer period. Therefore, the tests were performed on small scenarios to see what the system is capable of.

6.2 Energy savings

To determine energy savings that the system can provide, the system is compared to the worst case scenario and the current situation without the system running. The worst case scenario is leaving lights and the computer on always, this means a power consumption of 250 Watt during 8 hours which results in an power consumption of 2000 Wh.

The current situation in the office is that lights are turned of 10 minutes after leaving the office, which was determined by walking out the office and afterwards look back at the data from the Plugwise adapter. The PC turns of the monitor after 5 minutes and turns itself to sleep after 15 minutes, which is determined by looking at the PC's energy plan settings. A scenario of a typical workday is constructed the following way: two coffee breaks of 15 minutes, a lunch break of 30 minutes, and a meeting of 1 hour. In the current situation and given the typical scenario the power consumption in the office is 1675 Wh.

The proposed system was able to detect the absence of persons in the office within one minute. The lights are turned off after that minute and the PC is put to sleep resulting an power consumption, given the typical scenario, of 1536 Wh.

This is a decrease in power consumption of 8.3% given the current situation and 23.6% given the worst case scenario. This is shown in figure 6 where the Watts are shown that on average are used given the scenario mentioned before.

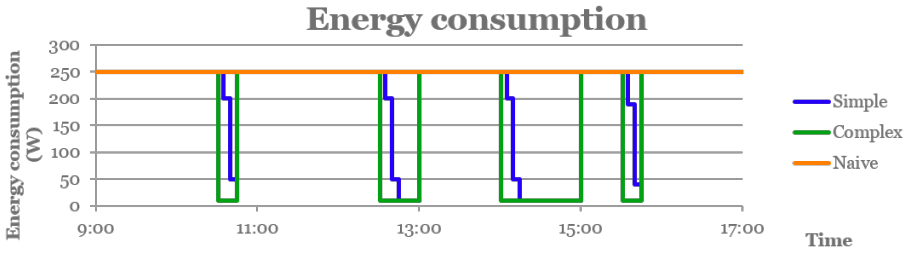

Figure 6: A graph showing potential energy consumption

6.3 Accuracy

The accuracy of the system is tested by using the shortcomings of the different sensors used in the test environment. The shortcoming of the presence (PIR) sensor is that it only detects changes in the infrared radiation reaching the sensor, which gives false data when someone is just sitting still at his desk. The shortcoming of the Sleepy for Linux activity sensor is that when someone is consuming data from his computer and not interacting with the computer, for instance when watching a video or reading a document, the activity data will turn up incorrectly. The shortcoming of the luminance (LUX) sensor is that people walking around the office can put more shadow over the LUX sensor resulting in a lower LUX level.

Given these shortcomings a set of scenarios is created to test the shortcomings:

- **Sitting still at the desk but typing on a computer**
  Most of the sensordata of the presence sensor results in a value of 15, which means no movement detected, and in on average one in 19 sensordata values the value indicates movement. However, the Sleepy for Linux activity sensor is detecting activity constantly because of the typing on the computer.

- **Watching a video and sitting still at the desk**
  Now both of the sensors have difficulties detecting that the user is present in the office. This was the only time the system turned the lights off while users were present in the office.

- **Watching a video while not sitting still**
  At this scenario the person watching the video moved a little bit during the video like drinking some water. The presence sensor now did detect movement on average in one of five sensordata points movement, which is enough to determine presence in the office.

- **Using the computer while no lighting is needed**
  In this scenario the computer is used in an office with enough natural light. Therefore the lighting doesn't have to be turned on.

  Due to the nature of the test environment, lighting was always needed. To test if how the system would cope with the situation, we lowered the threshold for the luminance sensor. The system coped well with this situation, it turned off the lights in the office and left the computer on.

7 Conclusion and discussion

From our theoretical and practical tests it is possible to answer the two main research questions.

The graph in figure 6 on page 26 shows that it would indeed be possible to save more energy by using this new system. It gave an energy reduction of 8.3% on top of the already existing energy savings. The program itself is not at all processor intensive. Though the scalability of the system has yet to be tested, one instance of this could easily be capable of controlling many offices at a time. Also, the current program is estimated to be able to run on device that is as energy efficient as Raspberry Pi 2 model B, which uses a maximum of 5W. Therefore it is safe to say that the benefits outweigh the cost of running the system.

As for the second research question, the tests have shown that certain situations where a single type of sensor was not sufficient have indeed shown an improved accuracy when using multiple sensors. One example is when you combine a PIR sensor with the sleepy activity sensor.

There was limited time and a limited test environment for this research to determine the full potential of the system. The system is tested with three sensors and two adapters, which is sufficient to test the correct working of the program and to get some energy saving indication. So testing the software in a wider test environment is a logical next step.

Other sensors like camera's with tracking capabilities could also be used to detect if someone is sitting behind his computer or is busy somewhere else in the room. This could improve the accuracy of the lighting, but it could especially improve the energy consumption for the computers in the office. Having such specific sensors, as camera's are, the result could be that users feel that they are watched too much. This is however not in the scope of this project and might be tested in further research.

Though it was beyond the scope of our testing environment, the adapters are not limited to lighting and computers. HVAC systems might also benefit from the availabilities of more sensors. Like the sensor network proposed in the research of Ahmadi et al.[1], that used sensors to detect temperature, humidity, and CO_2 levels to control a HVAC system.

It should be noted that the accuracy is highly dependent on the time between gathering the data points by the controller. When the amount of time is increased, it does reduce the processor intensiveness of the program, but it also introduces a delayed response. For the most favorable behaviour it is recommended to make this time between gathering data points less than 10 seconds, preferably between 1 and 5 seconds.

All in all, it can be concluded that the system has a high potential for saving energy and increasing the accuracy of existing automated power regulators. The nature of our system, a multiple sensor rule based environment, also opens up the pathway to usage beyond the field of energy reduction.

Home automation can also benefit from a centralized multiple-sensor based system. It can be used to automatically draw the curtains, start the dishwasher, plan and regulate the oven, automate the lighting and heating, etcetera.

It could also help elderly people with dementia. A door sensor could be used to notify those responsible if someone left their home during the night. A gas detector can be used for the stove to signal if the valve was left open.

There are however difficulties when it comes to sensor dependency. It might be that data from one sensor should influence the threshold of another sensor. There is not a continuous way that this can be done with the system. It can be done discreetly by creating more checker groups.

For instance the perceived temperature is dependent on both the humidity and the actual temperature. If you want to regulate this perceived temperature you'd want a temperature sensor threshold that is dependent on the value of the humidity sensor value. It would be possible to create multiple groups with certain humidity ranges and have different temperature thresholds for these ranges. This would solve the problem by using discrete ranges, but some accuracy would be lost compared to using the continuous humidity value.

8 Future work

8.1 Machine learning

At the moment the configuration file specifies the threshold for the sensors. These can be manually changed at any time (it does require a restart of the controller) and can be set to any value the user wants it to be. The problem is that the user might not always know what a good value could be for these thresholds. An addition to the system could be the use of machine learning to determine if a threshold value should be corrected from its originally specified value.

8.2 Support for different database systems

Currently the program uses the Cassandra database system and an REST interface on top of that to access the data. The program could be adapted to be more flexible with respect to retrieving data. For instance a non REST interface to retrieve data. Also the data representation could be more diverse, so instead of using only JSON other data representation languages could be used.

8.3 More general adapters

In the program only activate and deactivate is used to control devices, which could be in some situations and for some devices a bottleneck. So the software could be extended with more general adapters that can be told to which state a device has to go, for instance when using lights that can be dimmed, you probably don't want to fully turn on the light or fully turn off the light but a state in between is more energy-efficient or more user-friendly.

8.4 More than only energy saving

The system could be extended to be a more general system that can be used not only for energy saving but for controlling all kinds of devices not only focused on energy saving and makes lives of people easier. Such an extension would be more likely to be used outside office environments and more in homes. This is quite similar to the related work stated before of Google Nest Learning Thermostat and Homey, which are both aimed for the consumer market instead of for office environments.

8.5 User Interface

The program is just a simple command line executable, that has a configuration in the form of YAML file. This is relatively easy to edit, but with large system the configuration file becomes too large to get an easy overview of the system. Therefore, it could be useful to add a clear user interface which is able to configure the whole system in for instance a drag and drop kind of way to let users easily choose sensors instead of looking up UUIDs for that sensor.

Also statistics could be added to such an user interface, which helps people to get more insight in their behaviour and see their energy usage and the decision the program took to control devices.

References

[1] S.A. Ahmadi et al. “Towards more Efficient Building Energy Management Systems”. In: *Knowledge, Information and Creativity Support Systems* 1 (Nov. 2012), 118–125. DOI: {10.1109/KICSS.2012.21}.

[2] *Base64 encoding*. July 2015. URL: http://www.ietf.org/rfc/rfc1421.txt.

[3] Moonok Choi, Wan-Ki Park, and Ilwoo Lee. “Smart office energy management system using bluetooth low energy based beacons and a mobile app”. English. In: *Consumer Electronics* (Jan. 2015), 501–502.

[4] *EcoBee3 Smart Thermostat*. June 2015. URL: https://www.ecobee.com/.

[5] I. Georgievski, Tuan Anh Nguyen, and M. Aiello. “Combining Activity Recognition and AI Planning for Energy-Saving Offices”. English. In: *Ubiquitous Intelligence and Computing* (Dec. 2013), 238–245.

[6] *Google Nest Learning Thermostat*. June 2015. URL: http://nest.com/.

[7] *Gson Java Library*. July 2015. URL: https://github.com/google/gson.

[8] T. Hoeksema and M. Medema. *Sleepy For Linux*. June 2015. URL: http://sleepyforlinux.nl/.

[9] *Homey*. June 2015. URL: http://athom.com/.

[10] *Honeywell Lyric*. June 2015. URL: http://lyric.honeywell.com/.

[11] *Javascript Object Notation*. July 2015. URL: http://www.ietf.org/rfc/rfc4627.txt.

[12] Y. Kaneko et al. “An energy-saving office lighting control system linked to employee’s entry/exist”. English. In: *Consumer Electronics (GCCE)* (Oct. 2013), 440–444.

[13] Perez-Lombard L., Ortiz J., and Pout P. “A review on buildings energy consumption”. In: *Energy Build* 40.3 (2008), 394–398. DOI: {10.1016/j.enbuild.2007.03.007}.

[14] M. Magno et al. “A Low Cost, Highly Scalable Wireless Sensor Network Solution to Achieve Smart LED Light Control for Green Buildings”. English. In: *Sensors Journal* 15.5 (May 2014), 2963–2973.

[15] *RabbitMQ*. June 2015. URL: https://www.rabbitmq.com/.

[16] R. Saidur. “Energy consumption, energy savings, and emission analysis in Malaysian office buildings”. English. In: *ENERGY POLICY* 37.10 (Oct. 2009), 4104–4113.

[17] B. Setz and R Pul. *Lazy Sleep*. July 2013. URL: http://www.cs.rug.nl/~faris/students/Brian_Ruurtjan_report.pdf.

[18] *Universally unique identifier*. July 2015. URL: http://www.ietf.org/rfc/rfc4122.txt.

[19] *YAML Ain’t Markup Language*. July 2015. URL: http://yaml.org/spec/.