KIIIB

Andreas Moller s042809, David Emil Lemvigh
s042809

Kongens Lyngby 2011
IMM-PhD-2011-????

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-PhD-2011-????

Summary (English)

The goal of the thesis is to ...

Summary (Danish)

Målet for denne afhandling er at ...

Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark in fulfilment of the requirements for acquiring an M.Sc. in Informatics.

The thesis deals with ...

The thesis consists of ...

Lyngby, 01-January-2011

Andreas Moller s042809, David Emil Lemvigh s042809

Acknowledgements

I would like to thank my....

Contents

Introduction

this is the the intro baby

0.1 Scope

Define Context

- Present the domain of the project
- From he general to the specific
  - Distributed systems -> wireless -> ubiquitous computing
  - Include references to demonstrate you know the domain
- Identify important properties of the project domain
- What makes this project domain special?
  - Why do we need to undertake this project
- Define the scope of the project
- what sub-problems will be considered
- what sub-problems will *not* be considered

0.2 Goals

Identify Goals

- Project goals
- What are the main problems that the project should solve?
- Non-functional requirements
- Usability, performance, flexibility, extensibility, simplicity, security...
- Non-project goals
- experiments with (specific) new technology
- personal development

short answers

- Project goals
  - Create a home control system that relies on machine learning instead of programmability.
- Non-functional requirements
- Usability, performance, flexibility, extensibility, simplicity, security...
- Non-project goals
  - examine the possibilities of a system with minimal knowledge about its operating environment.
  - examine the possibilities of a system with minimal setup, and programming required

0.3 writing a good introduction

- what is the problem
  - explain the context of the project and identify goals

- what is the proposed solution?
  - Analyse problems and present proposed solution
- How do we evaluate the proposed solution?
  - Present evaluation criteria for proposed solution
  - define the method and scope of the project
- What did we learn?
  - Present a summary of the overall conclusions
  - lure the reader into reading the rest of the report

We want to create an intelligent smarthouse system, with a minimal setup requirement, to control the lighting. A system which after switches and sensors are physically installed, doesn't require further setup or technical expertise from the user. The system shouldn't require any direct user interaction.

The system should be able to gradually take over turning on and off the light. It should learn solely based on the user's normal behavior, which patterns leads up to turning the light on or off, for how long should the light stay on.

Ideally this should lead to increased comfort for the users, by not having to manually control the light. Possibly also reduced energy consumption, by only having light on when the users need it. Also by being more active about turning light off where it isn't needed, where most users wouldn't bother to get up to turn it off.

<this is not a finished product, but a proof of concept>

user interaction

The system should not require any direct interaction from the users. There has been a lot of change in software development the last few years, and on of the major focus areas has been on user interface design. The general theme is ease of use. You could say that the role of software is moving more towards being a servant, that a tool for the private user[^need-ref].

There has been a lot of change in software development the last few years, and one of the major focus areas has been on user interface design. The general theme is ease of use. You could say that the role of software is moving more towards being more a servant, than a tool for the private user [^need-ref]. elaborate on development in user interfaces [^need-ref]: Reference needed!

0.4 Project focus , assumptions and scope

<change the tone of the section. define focus areas instead of limitations >

Before we start to analyze what makes a good home control system we will outline which limitations, we will impose on the system. Some of these limitations are born out of personal interest<bad formulation!> , and some are there to limit the scope of the project.

To limit the scope of the project, we focus on controlling the lighting. This means that we will not go into appliances or house features such as heating, ventilation, or hot water. These are all areas where the system will be equally beneficial . However, modifying the system to incorporate these elements, would not require any major changes to the fundamental design of the core system. It would rather require a lot of time consuming implementation work.

We assume that people who have home control systems don't have windows . At the very least we will work under the assumption that the lighting conditions outside a room does not affect the conditions inside. A room with the lights turned on is light, and a room with the lights turned off is dark. This is simply to eliminate outside factors such as varying lighting conditions.Normally if there is enough natural lighting in a room, the user will not want the artificial light turned on.

The people in the house is trapped in "Groundhog Day" (**?**), or more precisely, Groundhog hour. We will assume that the users behavior and requirements does not change depending on date nor time of day. <TODO:Do we assume this.. or can we later explain that this assumption is unneccesarry!!???>

Finally we will make the assumption that a user only needs light in rooms where he or she is present. In reality situations may arise where the user would want the light to stay on when leaving at room, e.g., if he has a birdcage or an aquarium in the room.

<Only one user home!>

Analysis

The vision for our project is to create an intelligent home control system that requires minimum configuration and installation while providing optimal power saving, and comfort.

<fortsæt fra vision, og introduktion>

0.5 Smart House Survey

"If I have seen further it is by standing on the shoulders of giants" – Isaac Newton

The beginning of any good project starts with a survey of what already exists in the field. Which smart house solutions already exist, and what are their capabilities? What are the industry standards, if any? This section will provide a representative selection of smart house solutions. However, it will not be an exhaustive survey of all smart house solutions.

First we will establish some basic classifications of smart houses, to better compare the different systems. All systems can contain switches, sensors and remote controls, the difference is the functionally they provide, and how they operate.

We distinguish between three types of systems, which are derived from the taxonomies presented in Boguslaw Pilich's Master Thesis (**?**) :

(**?**): Boguslaw Pilich. Engineering Smart Houses, DTU IMM MSc Thesis Nr. 49/2004

0.5.1 Controllable houses

These are the simplest of the smart house solutions. Input devices like switches, remotes and sensors, can be setup to control output devices like appliance and dimmer switches, HVAC (Heating, Ventilation and Air Conditioning), etc. These solution may also include macros, e.g. where a single button may turn off all the lights in the home.

0.5.2 Programmable houses

These solutions incorporate some degree of logical operations, like having motions sensors not turn on the lights, if lux sensors are above a threshold. They may be able to have scheduled task, e.g. turning down the thermostats during stadard work-hours. The behavior of these systems have to be programmed by the manufacturer or the users. Consequetly, changes in user needs require the system to be reprogrammed.

0.5.3 Intelligent houses

In these solutions some form of artificial intelligence or AI is able to control the home. In computer science the term AI is used very loosely. I our case we will define an intelligent house, as a system that is capable of machine learning. That means that the system is capable of evolving behavioral patterns based on empirical data (1). Consequently, the system will over time adapt itself to changes in user needs.

The solutions presented, are some of the most widespread smart house solutions, and represent the three different types of systems: Controllable, Programmable and Intellight houses.

<TODO hurtig conclusion om hver løsning, hvad synes vi om dem>

INSTEON

INSTEON is a controllable home control system, targeted at private homes.

Nodes in the network can communicate using either RF signals or home's existing electrical wiring. A standard array of devices are supported:

- Dimmers & switches
- HVAC
- sprinklers
- motion sensors
- assorted bridge devices

INSTEON supports external application to be run on PC connected through a bridge devices to the network. By this logic it is technically possible to make the system programmable or even intelligent. However no commercial products providing these features currently exists. (2)

INSTEON's solution is fairly widespread in the US, and is a successor to the Redoak X10 system, and is compatable with it's product. It represents what a commercial controllable smart house is capable. It's functionaly very simplistic, but being able to communicate using the home electrical wiring, makes it a very non-intrusive system to install in an existing home.

Clipsal C-Bus

Clipsal is targeted at large scale home control. The system is install in such prominent buildings as the Sydney Opera house, Wembly Stadium and many more. Nodes communicate over it's own separate wired network, using the C-bus protocol. Each node has it's own microprocessor, which allows for distributed intelligence. Each node can also be individually programmed, and communicate over the shared bus. This allows unconventional devices like motors for stadium roofs and many other devices to be part of the network.

Clipsal's C-bus represents the flexibility and scalability programmable solutions on the market are able to achieve. It can also be installed in a private home. However, compared to other systems the rquirement of a separate wiring through-out a home can be a disadvantage. (3)

LK IHC

LK IHC is targeted at private homes. It can be installed with a wired network, or using wireless communication. This solusion tends to be build around simple switches, but with programmable scenarios, e.g. having a switch near the front

door and the master bedroom that turns off all lights. It is a modular system, where modules like wireless communication or alarms, can be added to the base installation.

The basic LK IHC installation is a controllable system. However, the modules can provide programmable functionality to the system, i.e. motion sensors normally control the lights, but if the alarm system is activated, the system calls 911. LK IHC was per 2008 installed in nearly 30% of newly constructed building in denmark. (4) (5)

MIT House_n

House_n differs from the previous systems, as it isn't a finished implementation, but a framework for research projects. There aren't any widespread commercially available intelligent smart house solution on the market, or atleast according to our classification of intelligent.

House_n represent one of many smart environment, build by universities around the world. The smart environments are homes for one or more inhabitants, and are part of a living laboratory. The living lab part of House_n is called PlaceLab, and is a one-bedroom condominium, inhabited by volunteers for varying lengths of time. These homes are designed for multi-disciplinary studies, of people and their pattens and interactions with new technology and smart home environments. Being university run smart homes, the work comming out of these facilities tends to be proof of concepts. (6)

<TODO men fair nok, de er proof on concept, hvilke projekter findes der. Eller hvorfor snakker vi ikke om dem>

The projects shown in this survey represent the solutions currently available or in development. There are many different controllable and programmable solution commercially available, with INSTEON, Clipsal C-bus and LK IHC being some of the more widespread representative solutions. INSTEON being a simple controllable solution, Clipsal C-bus and LK IHK are both programmable smart house solutions, but where LK IHC is designed for private homes, the Clipsal C-bus system is better suited for larger buildings.

MIT's House_n in this survey represent that truly intelligent smart houses only exists in demonstration environments and as proofs of concept, and are not yet widely available on the commercial market.

One of the main problems with current home control solutions is that installing such a system is rather costly and requires installation and configuration, which is rarely trivial . Some of the more advanced systems on the market, such as

the LK IHC, incorporate motion sensors and timers that automatically turn on and off lights or various appliances. These systems will save money over time, but they require extensive configuration or programming in order to function properly.

0.6 BIIIB

Of all the qualities mentioned in our vision for the system, power saving is the most important. **insert data on power consumption of average household** As seen in the survey above, this is an area where most modern home control systems falls short . Most systems are capable of providing only a modest reduction in power consumption, and some even increase the net consumption by adding the cost of running the control system. We want our system to differ from others on this specific aspect. In our system, reducing power consumption is the number one priority.

We will accomplish this by creating a system that focuses on turning off all lights and appliances where they are not needed. <what are the alternatives? why do we want to focus on this?>There are several advantages to this approach, compared to attempting to reduce the power consumption of active appliances. The main advantage is that it provides the largest reduction in power consumption. Most people remember to turn off the light in the bathroom, when they leave it, but this is far less common for the kitchen, or dining room, and only the most environmentally conscious people would ever turn off the light in the living room when they got to the bathroom. This means that there is a lot of wasted energy in the normal household .

An other advantage is that it incorporates perfectly with all other power reducing technologies. Bying appliances that use less energy will still give you the same percentage of power reduction as in a normal house.

This system will also eliminate the common problem of standby mode on many appliances such as tvs or stereos by having the appliance only in standby mode, when the user is likely to turn it on. The rest of the time the appliance is simply turned off.

Our approach to creating a house that is capable of predicting what the users want it to do, is to learn from what the user does and mimic these actions at the right times. To accomplice this, the system must do three things:

- The system must gather data on the users and their behavior in the house

- The system must analyze the data in order to build a decision scheme on which it will base its actions
- The system must be able control the house in real time, based on the decision scheme.

Since the system bases its decisions on data gathered on the user, the system is essentially trying to mimic useractions at the right times. The system will have three stages : <must the stages be introduced now? can they wait till the design section> * The untrained stage where the system is running, but it hasn't yet collected enough data to make intelligent decisions.

- The learning stage where there's enough data to attempt to manipulate the switches of the home. We call this the learning stage, because it provides us with a unique opportunity for the system to learn from the user. If the system makes a mistake and the user corrects it, e.g., the system turns off the lights and the user turns it back on, we can use that interaction to train our system further. In this case we can see it as the user punishing the system for making a mistake. The system will then adjust its decision scheme.
- After the system has been in the learning stage, it will enter its final stage, which we call the evolution stage. Here the system constantly updates its decision scheme with new data both from monitoring the user, and from being punished for its mistakes. In this stage there is a symbioses between the user and the system where the system reacts to the user and vice versa. <Som jeg forstår dette, så adskiller det sig ikke fra learning stage. Tekstmæssigt er forskellen, at her er det også brugeren, der ændrer opførsel. Men det har vel ikke noget med systemets udvikling at gøre. - men sandsynligt.>

0.7 Gathering data on the users

To mimic user actions, the system must first gather information on how the user interacts with the house. Therefore the first question we must answer is: What data should we collect on our users? In order for the system to effectively take over the users direct interactions with the house, we need to know two things.

- What action needs to be done?

- When shall the action be done?

The first question can be answered by monitoring the users direct interactions with the house. Since we have limited our system to handle lighting, this means monitoring the users interactions with the light switches.

The second question is a lot more complex. We need to collect data that can help us determine if the conditions are right for performing a specific action. We could of cause quite literally look at the time the action is performed, and then use that as a trigger, but this requires that users follow a very specific schedule.

To get a more detailed picture of when an action is done, we must analyze it relative to what the user is doing at the time. Since we're focussing on lighting this can be done simply by tracking the users movements. Thereby we will determine when an action shall be done based on where the user is, and where he is heading.

Perhaps the most obvious way of accomplishing this is by using cctv cameras. Using visual analysis is the most effective way of monitoring the user, as it will provide us with vast amounts of data on what the user is doing. By for example installing a fisheye camera in every room and use motion tracking on the video data stream, we can determine exactly where the user is, and what he is doing. While this is probably the solution that provides us with the most precise and detailed data, it does have one problem. Installing cameras in every room of the users house is, in out opinion, an unnecessary invasion of the users privacy. Even if the video data is not stored in the system, the presence of cameras will give many people the feeling of being watched in their own homes.

An other approach would be to use a beacon worn by the user that sends out a digital signal. The system could then use multilateration to pinpoint the exact location of the user. The beacon could be attached to the users keychain, incorporated into his cellphone, or, our personal science fiction favorite, injected under his skin. Like the camera approach this solution also has very high precision in tracking the user through the house. However,besides the point that the user might not always carry his keys or cellphone around, the main issue with this solution is scalability of users. . Even though we limit the system to one user for now , we want a system that can be scaled to accommodate multiple users acting both and autonomously. Having to attach a beacon to every visitor coming into the house is gonna be an annoyance, and without it the house would not react to the visitor at all.

The solution we chose is to use motion sensors. While this solution does not provide nearly the same precision in determining the users location as using fish

eye cameras or multilateration, motion sensors does come with a range of other advantages. Motion sensor is a very cheap solution, compared to installing cctv cameras, and will be far less invasive on the user's privacy. The motion sensor solution will also work for any user in the house, and does not require the user to carry any beacon device like in the multilateration system.

The system could easily be expanded by several other types of sensors as well. E.g. pressure sensors in the furniture, so the system can determine if there is someone present, even when motion sensors do not register them. There are several other examples of sensor technologies that could be incorporated in the system. Some will be discussed in the section 'Future work'.

For the moment we want to use as few hardware components as possible. There are two reasons for this:

- We want to keep the system as simple as possible from the consumers perspective. That means a system with as few components as possible.
- Creating a system that analyses and mimics user behavior will have a lot of unknown variables that are hard to predict no matter how it is implemented. It will therefore be preferable to start out with a system that is stripped down to the bare necessaries and then add components as the need for them arises.

We have chosen not to inquire??? any information on the position of the motion sensors in the house.<REWRITE, and explain> This means that the system does not know where each sensor is positioned, nor which other sensors are in the same room as it (the sensor og system?). This does make analyzing the data a lot more complicated, but we want to stick with the idea of minimizing the installation and configuration. This way the installation process can be boiled down to putting up the sensors, plugging in the system, and pressing "Start".

Choosing to only monitor the light switches and using motion sensors to track the user greatly simplifies the data collection. Both the motion sensors and the switches generate events when they are triggered, and the system should simply store these events in a database.

An alternative to this is to have the system analyze the data live, which would eliminate the need to store the event data. With this approach we do not have to store the events in the system, which over time could amount to a considerable amount of data. The problem is that if we should choose to modify the algorithms that analyze the data, we would effectively loose everything the system has learned so far. By storing the raw event data we can always recalculate

a new decision scheme based on the collected data. This solution leaves us with a lot more options later on. The collection of data must still happen in real time. Since it is very important that the events are recorded exactly when they happen, the system must not stall in this process.

0.8 Analyzing the collected data

"If you torture data long enough, it will tell you what you want!" -Ronald Coase Hvorfra

Now that we have a lot of data on our users interactions with the house, we need to analyze the data in order for our systems AI to act on the collected data. To be more specific: We need to create a decision scheme that the AI can use as a base for its decision making.

This is the critical part of the system. Collecting data, and acting based on an existing scheme are bot relatively simple tasks, however, designing the scheme to act, based on collected data, is far more complicated.

The purpose of analyzing the data is to find which specific situations that require the system to perform an action. Since the system does not know which sensors are located near which switches, the system will have to learn these relations based on the data collected. The simplest solution would be to have the system learn which switches and which sensors are located in the same room, and then create a "link" between them so the motion sensors control the light. This would result in what we have named the silvan[1] system.

The silvan system is basically having a motion sensor turn on the light when triggered, and then have a timer turn off the light if the sensor is not triggered for a set amount of time. The main problem with this kind of system is that if the user does not trigger a motion sensor regularly, the light will turn off when the user is still in the room. This is commonly a problem in a room like the living room, where the user will likely spend an extended amount of time sitting still. This problem can be addressed by extending the light's timeout time.

However, this brings us to the second problem. If the user is merely passing by a sensor, the light will still be turned on for its full duration. This greatly reduces the effectiveness of the system from a power saving point of view.

A better solution is to attempt to identify the users behavior leading up to a

[1] Danish building material retail-chain.

switch event . Since the system only use motions sensors to track the users movements, these sensor events will form the basis for the data analysis. The system could simply look at what sensor was triggered right before a switch was activated, and the create a link between that sensor and the switch. This, however, would result in a system much like the silvan system described above.

If we instead look at a series of sensor events leading up to a switch event, we will get a much more complex picture of what the user is doing. Since the switches in the house are located in fixed positions around the house, these movement patterns should repeat themselves relatively often. The movement patterns that lead up to a switch being turned off, will most likely also differ from a pattern leading up to a switch being turned on, since the user will be either entering or exiting a room. Once we have analyzed the data and identified the movement patterns related to a switch event, we need to create a decision scheme that the system can base its decision making on. That means we have to organize the analyzed data in a way so we easily can look up a specific pattern, and see whether it should trigger a switch action.

At first the system will need to gather enough data to make a solid analysis (the untrained stage). The length of this period will vary depending on how the data is analyzed. Unlike data collection, analyzing the data does not have strict time constraints. Since the decision scheme will be based on data collected over an extended period of time, the system will not benefit from having the decision scheme updated in real time. As a result the time constraints on analyzing the data will be quite loose, and should not pose as a restriction on the system. <reevalueate!>

<the house should react to the user, and the user to the house>

0.9 Controlling the house

After we have collected and analyzed data the the final task is to have the system control the house in real time, using the decision scheme created from the analyzed data. The system must constantly monitor the user and attempt to match his movement pattern to those present in the decision scheme. As with data collection this has to happen in real time so the patterns are not corrupted.

0.10 Requirement specification

Based of the analysis above we can now form a requirement specification for the project. The system shall collect data using motion sensors and by monitoring switches. This data should be stored as it is collected and without being manipulated.

The system should first be in an unlearned stage. In this stage the system should use the collected data to analyze the users movement patterns, leading up to a switch event, in order to create a decision scheme that can help the system mimic the actions of the user . The system should then enter a learning stage, where the system attempts to interact with the switches and learns from the users reactions. <shall we remove references to the stages? when does it change state?> Finally the system should enter, and stay in, an evolution stage, where the system constantly learns both from the actions and reactions of the user. <Svært at se, hvad forskellen mellem lærings og evolutionsfasen er. Ud over at systemet konsekvent bliver klogere og klogere? >

Design

- Develop the overall software architecture
  - Identify major system components
  - Specify interactions between system components
  - Specify system components
    - interfaces
    - semantecs
- The Design section should allow a skilled programmer to implement the system

NOTES: the system has two steps. * first it analyzes collected user data. This is one way communication. The user acts and the system listens. this part is done, and have been tested. Training period. * Now the system evolves by adjusting to the user. The System still collects data, as in step one, but it now interacts with the house, and the user reacts to the system. The system evolves based on the reactions of the user. The system never stops evolving.

0.11 theory

0.12 Sensor data

In order to train the system, some sensor data is needed, but how to obtain it? A couple of options are available:

1. Physical data, setting up wireless motion sensors and switches in a home, and collect the sensor data in a database. 1a A full installation, where the wireless switches are able to turn the light on and off, and by extension also the system. 1b A placebo installation, where the wireless switches doesn't control the light, and merely collect training data.
2. Simulated data, sing a simulator to generate data.
3. Constructed data, manually or algorithmicly generated data.

Looking only at the quality of the data, the best would be to setup an entire house with switches controlling the light, and motion sensors in every room. However installing the system into an existing home would be difficult. The motion sensors and wall socket switches would be fairly easy, but most homes have ceiling with wires in the walls. This means a complete installation of wireless switches would be difficult and costly. A placebo solusion to the complete installation would be to have wireless wall socket switches next to all the actual switches. By have the users press the placebo switches and the actual switches when they use the light, it's possible to get the training data without having to permanently install the system into the home. It does come with the drawback that it will only be able to generate data for the untrained stage, since the system would not be able to manipulate the actual switches.

People are not robots, and while we are creatures of habit, our movement patterns do not run like clockworks. No matter how well we would generate training data using simulators, algorithms or any other artificial method, there would always be a doubt on how close to actual human behavior it actually is.

We chose to install a placebo system of wireless switches and sensors, to collect training data. This gives us the best quality training data, for the untrained stage of the system, without the expenses of installing operational wireless switches.

With the training data, we can then use a simulator to evaluate the training stage. In the training stage, there it doesn't take that much data, to evaluate

that the system is learning properly. The data from the simulator is good enough to simulate simple movement patterns, to see which lights go on or off, as a simulated user moves from room to room.

0.13 Event patterns

One thing is knowing where the user is, another where the user is headed. By also looking at the preceding interval leading up to an event, it's possible to match that against previously observed patterns, to estimate where the user might be headed.

To determine these pattern we try to make some rules about what to look for. If too long time passes between event, the event are probably not part of the same movement pattern. But what is too long time?

0.14 Zones

In many cases to cover an entire room with sensors, the sensors end up overlapping in some areas. This overlapping can be used to increase the precision of the sensors. If two sensors triggers shortly after each other, then the user is in the zone where the two sensors overlap. In cases where multiple sensors triggers at the same time, it can be seen as one zone event.

Take (Figure 1) as an example, of three sensors which overlap a bit, and three paths past the sensors a, b and c. The paths b and c should only be observed as zone events by the system. While path a should look something like 1, zone 1 & 2, 2, zone 2 & 3, 3. depending on the cooldown of the sensors each event may be multiple times in the pattern.

0.15 Switch and sensor correlation

It is beneficial to get a sense of which sensors are near which switches. And we have a lot of statistical data too look at. When a user turns a which on, it most likely because there isn't light where the user intends to be in the immediate future. So it is possible to get an idea of which sensors are near a which, by looking at the interval shortly after a switch is turned on.

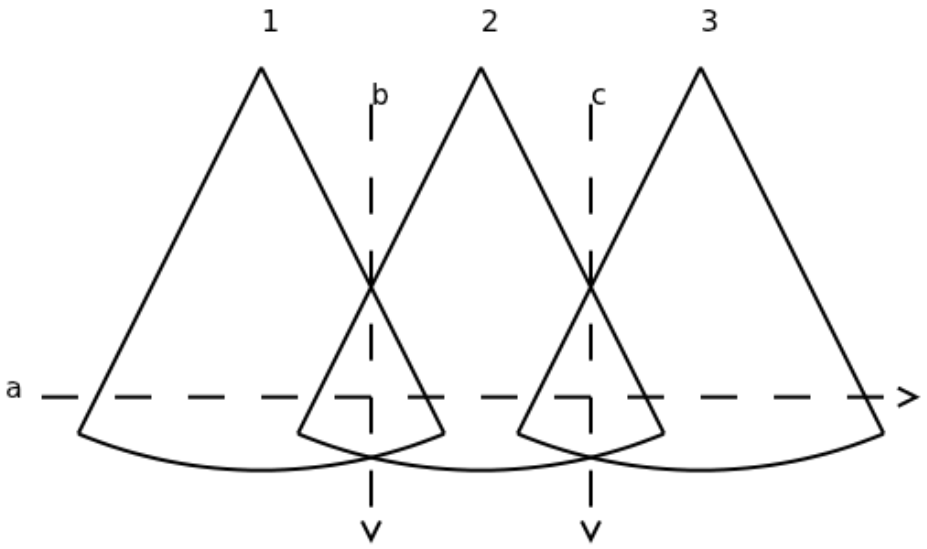


Figure 1: Sensors with overlapping zones

<TODO maybe talk about that is is less likely that a user will turn on a switch on, and then not enter that room>

When flicking a switch off, the user may be leaving the room, or just have entered the room to turn the switch off. Each of the two cases are just as likely as the other, but the sensor events in the interval leaving up to the off event is completely opposite.

<TODO you could possebly look at the interval after it's turned off, and say there are less likely to be in the room, and then try to reduce the correlation for those sensors (NYI)>

Based on the statistical data it is possible to generate a table of probability that a sensor is triggered shortly after a switch is turned on, and by extension of that give a idea of wich sensors are in the same room as a switch

$$P(sensor_i|switch_j, \Delta t) = \frac{\sum 1_{sensor_i}(switch_i, \Delta t)}{\sum switch_j \ events}$$

The indentity function $1_{sensor_i}(switch_i, \Delta t)$ is 1 if the sensor is triggered within Δt after $switch_j$ is triggered, and is not therefor not counted twice, in the sensor triggeres multiple times after the same switch event.

So to reiterate $P(sensor_i|switch_j, \Delta t)$ is the probability that $sensor_i$) fires within Δt after $switch_j$ fires.

Table 1: Correlation table

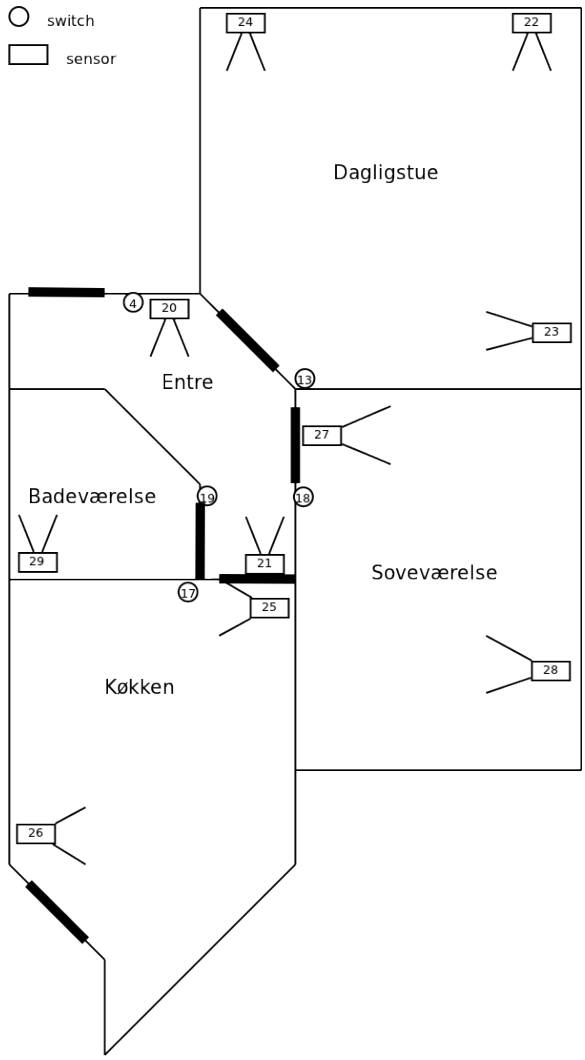
	sensor 1 (se_1)	sensor 2 (se_1)	...	sensor n (se_n)
switch 1 (sw_1)	$P(se_1\|sw_1, \Delta t)$	$P(se_2\|sw_1, \Delta t)$	...	$P(se_n\|sw_1, \Delta t)$
switch 2 (sw_2)	$P(se_1\|sw_2, \Delta t)$	$P(se_2\|sw_2, \Delta t)$	...	$P(se_n\|sw_2, \Delta t)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
switch m (sw_m)	$P(se_1\|sw_m, \Delta t)$	$P(se_2\|sw_m, \Delta t)$	...	$P(se_n\|sw_m, \Delta t)$

Implementation

- Description of the system components
- The implementation section should allow a skilled programmer to maintain the software
- **Remember, the most important documentation of the implementation is comments in the code!**

0.16 Training data collection

In order to collect training data, we installed wireless switches and PIR[ˆpir] sensors in a home (**??**). The placebo switches were placed next to the normal switches controlling the light for each room, in all cases being the switch closest to the entrance. Each room have one or more PIR sensors, averaging 2 per room, with one lest in the restroom and an additional sensor in the living room. When placing the sensors, the system can obviouly only laern from behavoir in areas covered by sensors. So sensors should provide as close to full coverage as possible, with special emphasis on making sure the entrances are covered.



[ˆpir] Passive infrared sensors.

The wireless nodes we have available communicate using the Zensus Z-Wave protocol. We setup a mini PC with a Z-Wave serial device, and configured all PIR sensor and switches to send notifications to the PC, when they where activated. The PC ran a Z-Wave API, which we added a listener to, so that sensor and switch event was logged to a SQL database.

Table 2: Database table for sensor events

sensor_events	
id	Integer
timestamp	Timestamp

Table 3: Database table for switch events

switch_events	
id	Integer
timestamp	Timestamp
status	Boolean

0.17 Simulator /AI interface

We have a smart house simulator available, which will be extended with an AI module, implementing the features discussed in this report. The simulator is implemented in scala, so an obvious choise would be to implement the AI in scala aswell. However work with the simulator in the initial stages of the project, showed that our programming speed in scala was too slow to get any meaningful amount of work done. The scala language is build upon Java, and both languages compiles to bytecode in *.class* files. A result of that is that Scala and Java interface very easily, and Scala code can invoke Java methods and vice versa. We chose to implement the AI in Java, working in a language we're well-versed in, to increase our productivity and quality of the code.

0.18 Decision table /matrix /markov /thingie...

Antallet af gange den klasse har skiftet navn... I've lost count... <TODO Andy, you deal with it>

0.19 Event patterns

To make lookups based on the lastest event pattern, each new sensor event needs to be a matched to see if it's part of a pattern. As each sensor and switch event

is received by the system, a list of the most recent event pattern is maintained in an EventList.

EventList determines if the lastest event is part of the pattern, and determines if a zone event has occured (if set to use zone events).

The invariant of the EventList, is that after an event is added, the event list contains the current event patttern. This pattern can then be used to determine if any switches should be turned on or off.

0.20 Correlation table

0.20.1 Correlation statistical generation

Correlation calculates the probability that a sensor is correlated a switch. It scans the database, and looks at the interval just after a switch is triggered. The sensors that triggered in the interval, are counted for that interval, in a way thay they're only counted once per switch event. If a multiple switch event are triggered in the same interval, the sensor events in the overlapping intervals should be counted for each of those switches. Having the number of times each switch is triggered, and each sensors triggeres with the given time interval, it's then calculated the probability that $sensor_i$ is triggered, given that a $switch_j$ was turned on atmost Δt ago. This gives the statistical correlation probability table.

To this the correlation confirmations in the database, is then added. Each row in the database table contains the accululated correlation correction for that switch /sensor pair. The correlation correction is simply added to the correlation based on the statistical data.

The resulting correlation table is allowed to have probabilities above 100%, which is inteded as destribed in

0.20.2 Correlation correction

When a switch is turned on, a timer is started for that switch. If a correlated sensor is triggered, it timer is extended. The duration is determined by the correlation between the sensor and the switch, higher correlation gives longer

timeouts. If the switch is turned off, the timer is stopped. If the timer runsout a timeoutevent is triggered, and the light is turned off, and a new timer is started, to verify that no manual overrides occur. If the a manual override occurs (e.g. the user turns the switch on again, while the timer is running), the system is "punished". The system increases the timeout time, by increasing the correlation between the switch and the first sensor triggered after the switch was turned off. If no manual override occurs, the system was correct in turning off the light, and lowers the timeout time, by reducing the correlation between switch and the last seen sensor before the switch was turned off.

These correlation corrections are stored in a separate table in the database. The correlation use for the timeout is based on both the statistical correlation, and these correlation corrections. The correlation corrections increase or reduce the correlation by 10 percent points. The system allows correlations higher than 100%, this gives the intended behavoir that a switch may have a longer timeout than what is default.

0.21 Timers and timeout

Timers are implemented in the Timer and Sleeper class. Sleeper is a fairly simple class, it sleeps starts a new thread, sleeps for a given time, then fires a timeout event to a given timeout listener. Timer simply holds a map, where each switch can set a timeout. Timer creates a sleeper object, and puts in the map. The sleepers can then easily be monitored and interrupted if needed.

To received the timeout events the SmartHouse class implements TimeoutListener.

Evaluation

"I have not failed. I've just found 10,000 ways that won't work." – Thomas A. Edison

"Laughing at our mistakes can lengthen our own life. Laughing at someone else's can shorten it." – Cullen Hightower

"To err is human–and to blame it on a computer is even more so." – Robert Orben

- Evaluation should document that the goals have been achieved
  - Functional requirements (i.e., testing)
  - Non-functional requirements (e.g., performance)
- Definition of the evaluation strategy
  - Qualitative-/quantitative evaluation
  - Software testing
    - white-box/black-box
    - testing levels *unit testing, integrationg testing, system testing, acceptance testing
- summarised output from the evaluation *output should be explained
  - provisional conclusions should be presented

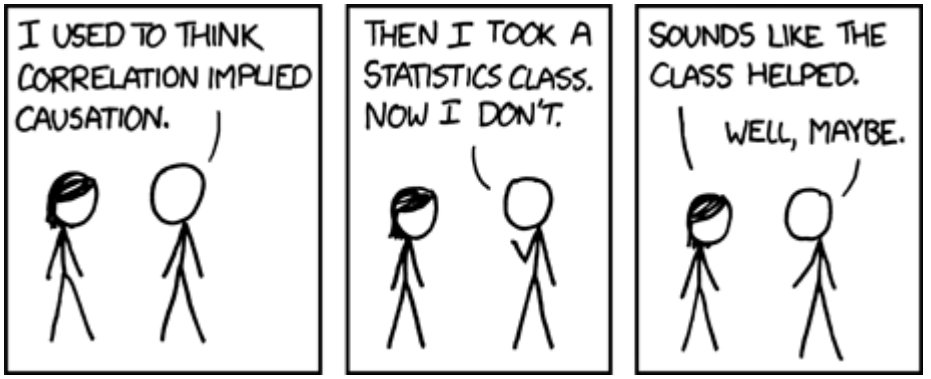


Figure 2: XKCD Correlation

0.22 Correlation

Before evaluating the correlation probability table, some goals should be established.

- A sensor should have the highest correlation to the switch in the room it's in.
- Ideally some threshold exists, so that all correlations above the threshold are in the same room, and all other correlations are below the threshold.

Table 4: Correlation table, based on statistical data. > 40% in bold, 40–20% in italic.

	Switches	Sensors									
		20	21	22	23	24	25	26	27	28	29
		Hallway		Living room			Kitchen		Bedroom		WC
4	Hallway	**0.4**	**0.67**	0	0.2	0.13	0.07	0	0	0.07	0
13	Living room	*0.35*	*0.23*	0.12	*0.27*	**0.42**	0.04	0.04	0.08	0.08	0
17	Kitchen	*0.22*	*0.28*	0	0.03	0.17	*0.39*	**0.58**	0.14	0.03	0.03
18	Bedroom	0.1	0.13	0	0	0.03	0.03	0	**0.57**	**0.6**	0.03
19	WC	*0.29*	*0.29*	0.06	0.09	0.08	0.06	0	0.07	0.03	**0.75**

The correlation table (Table 4) is based on collected data from the testing environment. The first criteria holds, that all sensors have the highest correlation with the switch in the room they're in. Most (but not all) the correlation probability between sensors and switches in the same room are above 40%. All

correlations between for switches and sensors not in the same room are below 40%. Three sensors have correlations lower than 40% to the switch in the room they're in, and one of them as low as 12%. Two of the three sensors in the living room, not only have correlations below 40%, but correlations below those of sensors in the adjecent hallway. As can be seen in the overview of the appartment (**??**), the sensors 22 and 25 are located in the far end of the rooms from the switch and doorway. Since the calculated correlation probabilities are based on the time interval right after the light is turned on, it makes sense that these sensor, relatively far away from the switches ends up with a lower correlation.

Sensor 23 is a bit more interesting, since it located fairly close to the doorway. Having one of the authors of this thesis also being the guinea pig running around generating sensor data, gives a unique insight why some sensor patterns look the way they do. Sensor 24 is located by a desk, and 23 next to a sofa. So in this case different user activities triggeres different sensor, in this case sitting on the sofa and watch TV, or go the the desk and work.

One thing to note is, these are the probabilities based solely on the statistical data, and that correlation corretions would be added onto this schema. So it doesn't perfectly reflect the room /switch + sensor correlations on it's own, though it gives a close approximation

Conclusion

- summarises all the result of the project
  - what was the problem?
  - what has been achieved?
- presents final conclusions *summary of provisional conclusions
  - further conclusions drawn from the sum of evidence
- presents directions for future work
  - new problems identified through the project
  - outline the possible evolution curve of the software

0.23 writing good conclusions

- What was the problem?
  - Remind the reader of the context and project goals
- What was the proposed solution? -Remind the reader of the proposed solution -what was done in the project
- How did we evaluate the proposed solution? -Summarize results of indicidual experiments. -this inclydes any testing of software in development projects -Draw conclusions on the endicidual experiments

- What did we learn? -Present overall conclusions of the project
  - Outline ideas for future work

0.24 Awesome quotes

lille liste af citater som godt kunne passe ind i rapporten

I am among those who think that science has great beauty. A scientist in his laboratory is not only a technician: he is also a child placed before natural phenomena which impress him like a fairy tale. – Marie Curie

Good judgment comes from experience, and experience comes from bad judgment. – Barry LePatner

Nothing can be so amusingly arrogant as a young man who has just discovered an old idea and thinks it is his own. – Sidney J. Harris

The more original a discovery, the more obvious it seems afterwards. – Arthur Koestler

Bibliography

[1] Wikipedia article on machine learning. http://en.wikipedia.org/wiki/Machine_learning

[2] INSTEON. http://www.insteon.net

[3] Wipedia article on the Clipsal C-Bus protocol. http://en.wikipedia.org/wiki/C-Bus_(protocol)

[4] Mads Ingwar and Soeren Kristian Jensen. IMM Smart House Project: a state of the art survey. 2008.

[5] Lauritz Knudsens. http://www.lk.dk

[6] MIT House_n. http://architecture.mit.edu/house_n/placelab.html

.1 Source Listings

.1.1 Package: smarthouse

.1.1.1 SmartHouse.java

```
1  package smarthouse;
2
3  import java.sql.DriverManager;
4  import java.sql.SQLException;
5  import java.sql.Connection;
6  import java.sql.Statement;
7  import java.util.ArrayList;
8  import java.util.HashMap;
9  import java.util.List;
10 import java.util.Map;
11
12 import timer.TimeoutEvent;
13 import timer.TimeoutListener;
14 import timer.Timer;
15
16 import events.*;
17 import config.Config;
18 import core.*;
19
20 public class SmartHouse implements TimeoutListener {
21
22   private static boolean debug = true;
23   Connection conn = null;
24   Statement stmt;
25   AI ai;
26   EventList eventlist,zoneeventlist;
27   Correlation correlation;
28   Timer timer;
29   List<Integer> timeout;
30   int onTime;
31   int punishmentTimeout;
32   Map<Integer, Boolean> switchStatus;
33   Map<Integer, Integer> firstSensorAfterTimeout;
34   DecisionMatrix decisionMatrix;
35   public static void main(String[] args){
36     SmartHouse sh = new SmartHouse();
37   }
38
39   /*
40    * Constructor for the class SmartHouse
41    * Handles the input and output for the ai
42    */
43   public SmartHouse(){
44     Config.loadConfig();
45     try {
46       debug = Config.debug;
47       Class.forName("com.mysql.jdbc.Driver");//load the mysql driver
48       conn = DriverManager.getConnection("jdbc:mysql://localhost/kiiib?
             user=KIIIB&password=42");//connect to the database
49       stmt = conn.createStatement();
50       decisionMatrix = new DecisionMatrix();
51       correlation = new Correlation();
52
53       eventlist = new EventList();
```

```
54        zoneeventlist = new EventList(true);
55        timer = new Timer();
56        timeout = new ArrayList<Integer>(10);
57        onTime = Config.defaultOnTime;
58        punishmentTimeout = Config.punishmentTimeout;
59        firstSensorAfterTimeout = new HashMap<Integer, Integer>();
60        switchStatus = new HashMap<Integer, Boolean>();
61        for (int sw : decisionMatrix.switches){
62          switchStatus.put(sw,false);
63        }
64      }
65      catch (SQLException se){
66        System.out.println("SQLException: " + se.getMessage());
67        System.out.println("SQLState: " + se.getSQLState());
68        System.out.println("VendorError: " + se.getErrorCode());
69
70      }
71      catch (Exception e){
72         e.printStackTrace();
73       }
74    }
75
76    public SmartHouse(AI ai) {
77      this();
78      this.ai = ai;
79    }
80
81    /*
82     * Method called when a sensorevent occurs in the simulator
83     * @author Andreas m ller & David Emil Lemvigh
84     */
85    public void sensorEvent(int sensorId){
86      try{
87        System.out.println("Sensor "+sensorId+" fired!");
88        eventlist.sensorEvent(sensorId);
89        zoneeventlist.sensorEvent(sensorId);
90
91        if (!debug)
92          stmt.executeUpdate("INSERT INTO sensor_events VALUES("+sensorId+
               ",NOW())");
93
94        for (int sw : timeout) {
95          if (!firstSensorAfterTimeout.containsKey(sw))
96            firstSensorAfterTimeout.put(sw, sensorId);
97        }
98        for (int sw : correlation.getSwitches(sensorId, 0.5f)) {
99          if (isOn(sw) && !timeout.contains(sw)) {
100           float t = onTime * correlation.getCorrelation(sw, sensorId);
101           System.out.printf("keep switch %d on (%d ms)\n", sw, (long) t)
                  ;
102           timer.updateTimeout(sw, (long) t, this);
103         }
104       }
105     }
106     catch(SQLException se){
107       System.out.println("SQLException: " + se.getMessage());
108       System.out.println("SQLState: " + se.getSQLState());
109       System.out.println("VendorError: " + se.getErrorCode());
110     }
111     matrixLookUp();
112   }
113   /*
114    * Method called when a switch event occurs in the simulator
115    * @author Andreas M ller & David Emil Lemvigh
116    */
```

```
117   public void switchEvent(int switchId, int status){
118     try{
119       System.out.println("Switch "+switchId+" turned "+status);
120   //              System.out.println(eventlist);
121       boolean cmd = (status == 1) ? true : false;
122
123       if (cmd) {
124         if (timeout.contains(switchId)) {
125           timeout.remove((Object) switchId);
126           timer.stop(switchId);
127           if (firstSensorAfterTimeout.containsKey(switchId))
128             correlation.increaseCorrelation(switchId,
                    firstSensorAfterTimeout.get(switchId));
129         }
130         on(switchId);
131         timer.setTimeout(switchId, onTime, this);
132
133       } else {
134         off(switchId);
135       }
136       if (!debug)
137         stmt.executeUpdate("INSERT INTO switch_events VALUES("+switchId+
                ","+status+",NOW())");
138     }
139     catch(SQLException se){
140       System.out.println("SQLException: " + se.getMessage());
141       System.out.println("SQLState: " + se.getSQLState());
142       System.out.println("VendorError: " + se.getErrorCode());
143     }
144   }
145
146   private Map<Integer, Boolean> testMap = new HashMap<Integer, Boolean
          >();
147
148     public void TimeoutEventOccurred(TimeoutEvent event) {
149     System.out.println("I should probably turn off the light now");
150     int id = (Integer) event.getSource();
151     if (timeout.contains(id) && eventlist.getLastEvent() != null) {
152       correlation.reduceCorrelation(id, eventlist.getLastEvent().getID()
              );// adjust for zoneeventlist
153       timeout.remove(event.getSource());
154     } else {
155       off(id);
156       timeout.add(id);
157       timer.setTimeout(id, punishmentTimeout, this);
158     }
159   }
160   private void matrixLookUp(){
161     try{
162       KeyList keylist;
163       int P;
164       float value = 0;
165       for (int sw : decisionMatrix.switches){
166         keylist = new KeyList(eventlist);
167         keylist.add(sw);
168         if(switchStatus.get(sw)){
169           if(decisionMatrix.off.containsKey(keylist)){
170             value = decisionMatrix.off.get(keylist);
171           }
172           System.out.println("probability value : "+value);
173           if(value>Config.probabilityThreshold){
174             off(sw);
175           }
176           if(Config.useZones){
177             if(decisionMatrix.off.containsKey(keylist)){
```

```
178                 keylist = new KeyList(zoneeventlist);
179                 keylist.add(sw);
180                 value = decisionMatrix.off.get(keylist);
181               }
182
183             }
184
185           }
186           else{
187             if(decisionMatrix.on.containsKey(keylist)){
188
189               value = decisionMatrix.on.get(keylist);
190             }
191             if(Config.useZones){
192               if(decisionMatrix.on.containsKey(keylist)){
193                 keylist = new KeyList(zoneeventlist);
194                 keylist.add(sw);
195                 value = decisionMatrix.on.get(keylist);
196               }
197
198             }
199             System.out.println("probability value for switch  "+sw+" : "+
                    value);
200             if(value>Config.probabilityThreshold){
201               on(sw);
202             }
203
204           }
205         }
206       }
207       catch(Exception e){
208         e.printStackTrace();
209       }
210     }
211
212     private void on(int id) {
213       System.out.println("Turning switch "+id+" on");
214       ai.on(id);
215       switchStatus.put(id, true);
216     }
217
218     private void off(int id) {
219       System.out.println("Turning switch "+id+" off");
220       ai.off(id);
221       switchStatus.put(id, false);
222     }
223
224     private boolean isOn(int id) {
225       if (switchStatus.containsKey(id))
226         return switchStatus.get(id);
227
228       return false;
229     }
230   }
```

Listing 1: SmartHouse.java

.1.2 Package: timer

.1.2.1 Sleeper.java

```
1  package timer;
2
3  import javax.swing.event.EventListenerList;
4
5  public class Sleeper extends Thread {
6
7      private int id;
8      private long time;
9      private long end;
10     private TimeoutListener listener;
11
12     public static void main(String args[]) throws InterruptedException {
13         System.out.println("here we go...");
14         new Sleeper(1, 1000);
15         new Sleeper(2, 2000);
16         new Sleeper(2, 2000);
17         new Sleeper(3, 3000).join();
18         System.out.println("all done");
19     }
20
21     public Sleeper(int id, long time) {
22         this.id = id;
23         this.time = time;
24         this.end = System.currentTimeMillis() + time;
25         this.start();
26     }
27
28     public Sleeper(int id, long time, TimeoutListener l) {
29         this(id, time);
30         this.listener = l;
31     }
32
33     public long getEnd() {
34         return end;
35     }
36
37     public void run() {
38         try {
39             sleep(time);
40             System.out.println(id + ": done");
41
42             if (listener != null) {
43                 listener.TimeoutEventOccurred(new TimeoutEvent(id));
44                 System.out.println(id + ": event fired");
45             }
46         } catch (InterruptedException ex) {
47             return;
48         }
49     }
50 }
```

Listing 2: Sleeper.java

.1.2.2 Timer.java

```java
package timer;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import javax.swing.event.EventListenerList;

public class Timer implements TimeoutListener {


    private Map<Integer, Sleeper> timers;
    private TimeoutListener listener;

    public static void main(String[] args) throws Exception{
        Timer t = new Timer();
        t.setTimeout(1, 1000, t);
        t.setTimeout(2, 2000, t);
        t.setTimeout(3, 2000, t);
        Thread.sleep(1000);
        t.setTimeout(3, 2000, t);
    }

    public Timer() {
        timers = new HashMap<Integer, Sleeper>();
    }

    public Timer(TimeoutListener l) {
        this.listener = l;
    }

    public void setTimeout(int id, long time) {
        setTimeout(id, time, listener);
    }

    public void setTimeout(int id, long time, TimeoutListener l) {
        if (timers.containsKey(id))
            timers.get(id).interrupt();

        timers.put(id, new Sleeper(id, time, l));
    }

    /**
     * set the timeout, only if a timer is already is set for the id,
     * and the new timeout will end later than the old timeout
     * @param id
     * @param time
     */
    public void updateTimeout(int id, long time, TimeoutListener l) {
       if (!timers.containsKey(id) || !timers.get(id).isAlive())
           return;

       if (timers.get(id).getEnd() < System.currentTimeMillis() + time)
           setTimeout(id, time, l);
    }

    public void updateTimeout(int id, long time) {
        updateTimeout(id, time, listener);
    }

    public void stop(int id) {
        timers.get(id).interrupt();
    }

```

```
65      @Override
66      public void TimeoutEventOccurred(TimeoutEvent event) {
67          // TODO Auto-generated method stub
68          System.out.println(event.getSource() + ": event detected");
69      }
70
71  }
```

Listing 3: Timer.java

.1.2.3 TimeoutListener.java

```
1  package timer;
2
3  import java.util.EventListener;
4
5  public interface TimeoutListener extends EventListener {
6
7      public void TimeoutEventOccurred(TimeoutEvent event);
8
9  }
```

Listing 4: TimeoutListener.java

.1.2.4 TimeoutEvent.java

```
1  package timer;
2
3  import java.util.EventObject;
4
5  public class TimeoutEvent extends EventObject {
6
7      public TimeoutEvent(int id) {
8          super(id);
9      }
10
11 }
```

Listing 5: TimeoutEvent.java

.1.3 Package: events

.1.3.1 EventList.java

```
1  package events;
2
3  import java.util.HashSet;
4  import java.util.Iterator;
5  import java.util.LinkedList;
```

```
6
7  import config.Config;
8
9  public class EventList {
10
11     private LinkedList<Event> events;
12 //     private LinkedList<Event> zone;
13
14     /**
15      * Maximum interval between sensor events, for the event to be
            considered a zone event.
16      *  Default value 1 sec.
17      */
18     private int zone_interval;
19
20     /**
21      * Time interval stored in the event list.
22      */
23     private int pattern_interval;
24     private int pattern_length;
25     private boolean useZones = true;
26
27     public static void main(String[] args) {
28         EventList list = new EventList();
29         list.sensorEvent(1);
30         list.sensorEvent(2);
31         list.sensorEvent(3);
32         System.out.println(list);
33     }
34
35     public EventList() {
36         events = new LinkedList<Event>();
37 //         zone = new LinkedList<Event>();
38         this.pattern_interval = Config.patternInterval;
39         this.pattern_length = Config.patternLength;
40         this.zone_interval = Config.zoneInterval;
41         this.useZones = Config.useZones;
42     }
43
44     public EventList(boolean useZones) {
45         this();
46         this.useZones = useZones;
47     }
48
49     public EventList(int zone_interval, int pattern_interval, int
            pattern_length) {
50         this();
51         this.zone_interval = zone_interval;
52         this.pattern_interval = pattern_interval;
53         this.pattern_length = pattern_length;
54     }
55
56
57
58     /**
59      * Add event
60      * @param e
61      */
62     public void add(Event e) {
63         removeOld(e.getTS());
64
65         if(useZones && e instanceof SensorEvent)
66             determineZone(e);
67         else
68             events.add(e);
```

```
69
70          while (events.size() > pattern_length)
71              events.removeFirst();
72      }
73
74      /**
75       * removes all events if more than pattern interval has passed since
             the last event
76       * also mantains a maximum pattern depth
77       */
78      private void removeOld(long time) {
79          if(events.size() > 0 && time - events.getLast().getTS() >
                pattern_interval)
80              events.clear();
81
82      }
83
84      private int currentPatternLength() {
85          int count = 0;
86          for (Event e : events)
87              if (e instanceof SensorEvent || e instanceof ZoneEvent)
88                  count++;
89          return count;
90      }
91
92      private void determineZone(Event e) {
93          if (events.size() > 0 && events.getLast().getTS() +
                zone_interval > e.getTS()) {
94
95              Event last = events.getLast();
96              if (last instanceof ZoneEvent) {
97                  boolean contains = false;
98                  ZoneEvent z = (ZoneEvent) last;
99                  for (int id : z.getIDs()) {
100                     if (id == e.getID()) {
101                         contains = true;
102                         break;
103                     }
104                 }
105                 if (!contains) {
106                     z.addID(e.getID());
107                     return;
108                 }
109
110             } else if (last instanceof SensorEvent){
111                 if (last.getID() != e.getID()) {
112                     events.removeLast();
113                     events.addLast(new ZoneEvent(last.getTS(), last.
                            getID(), e.getID()));
114                     return;
115                 }
116             }
117         }
118         events.add(e);
119     }
120
121     public String toString() {
122         StringBuffer sb = new StringBuffer("==== Event list ====\n");
123         for (Event e : events) {
124             sb.append(e.toString() + "\n");
125         }
126         return sb.toString();
127     }
128
129     public void sensorEvent(int id) {
```

```
130             add(new SensorEvent(id));
131         }
132
133         public void switchEvent(int id, int status) {
134             boolean cmd = (status == 0) ? false : true;
135             add(new SwitchEvent(id, cmd));
136         }
137
138         /**
139          * get events in event list, including detected zone events
140          * @return
141          */
142         public Event[] getEvents() {
143             Event[] array = new Event[events.size()];
144             events.toArray(array);
145             return array;
146         }
147
148         public Event[] getDistinctEvents() {
149             HashSet<Event> set = new HashSet<Event>(events);
150             Event[] array = new Event[set.size()];
151             set.toArray(array);
152             return array;
153         }
154
155         /**
156          * get only sensor and zone events
157          * @return
158          */
159         public Event[] getPattern() {
160             Event[] pattern = new Event[pattern_length];
161             //if current pattern depth is less than pattern depth, fill
                    missing with -1
162             for (int i = 0; i < pattern_length - currentPatternLength(); i
                    ++) {
163                 pattern[i] = new SensorEvent(-1);
164             }
165
166             Iterator<Event> it = events.iterator();
167             for (int i = pattern_length - currentPatternLength(); i <
                    pattern_length; i++) {
168                 pattern[i] = it.next();
169             }
170             return pattern;
171         }
172
173         public Event getLastEvent() {
174             if (events.size() > 0)
175                 return events.getLast();
176
177             return null;
178         }
179       public boolean containsZoneEvent(){
180         if(useZones){
181           for(Event e : events){
182             if (e instanceof ZoneEvent)
183                 return true;
184           }
185         }
186         return false;
187       }
188     }
```

Listing 6: EventList.java

.1.3.2 Event.java

```
1  package events;
2
3  import java.text.SimpleDateFormat;
4  import java.util.Date;
5
6  public abstract class Event {
7
8      private static SimpleDateFormat sdm = new SimpleDateFormat("[HH:mm:
           ss]");
9
10     protected int id;
11     protected long ts;
12
13     public Event(int id, long ts) {
14         this.id = id;
15         this.ts = ts;
16     }
17
18     public Event(int id) {
19         this(id, System.currentTimeMillis());
20     }
21
22     public int getID() {
23         return id;
24     }
25
26     public long getTS() {
27         return ts;
28     }
29
30     public boolean compareID(int id) {
31         return this.id == id;
32     }
33     public boolean equals(Object o) {
34         if (!(o instanceof Event)) {
35             return false;
36         }
37         Event e = (Event) o;
38         if (e.id != this.id)
39             return false;
40         if (e.ts != this.ts)
41             return false;
42
43         return true;
44     }
45
46     public int hashCode() {
47         return id ^ (int) ts;
48     }
49
50     /**
51      * return timestamp as human readable string
52      * @return
53      */
54     public String tsString(){
55         return sdm.format(new Date(ts));
56     }
57
58 }
```

Listing 7: Event.java

.1.3.3 SensorEvent.java

```
package events;

public class SensorEvent extends Event {

    public SensorEvent(int id, long ts) {
        super(id, ts);
    }

    public SensorEvent(int id) {
        super(id);
    }

    public String toString() {
        return tsString() + " Sensor event " + this.id;
    }

    public boolean equals(Object o) {
        if (!super.equals(o))
            return false;

        if (!(o instanceof SensorEvent))
            return false;

        return true;
    }

}
```

Listing 8: SensorEvent.java

.1.3.4 ZoneEvent.java

```
package events;

import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;

public class ZoneEvent extends Event {

    protected int[] ids;

    public ZoneEvent(int ... ids) {
        super(0);
        Arrays.sort(ids);
        this.ids = ids;

        this.id = getID(ids);
    }

    public ZoneEvent(long ts, int ... ids) {
        this(ids);
        this.ts = ts;
    }

    public ZoneEvent(List<Event> zone) {
        this(zone, System.currentTimeMillis());
    }
```

```
27
28      public ZoneEvent(List<Event> zone, long ts) {
29          super(0);
30
31          ids = new int[zone.size()];
32          for(int i = 0; i < zone.size(); i++)
33              ids[i] = zone.get(i).getID();
34
35          Arrays.sort(ids);
36
37          this.id = getID(ids);
38          this.ts = zone.get(zone.size()-1).getTS();
39      }
40
41      private static int getID(int ...ids) {
42          int sum = 0;
43          for (int i : ids)
44              sum = sum*256 + i;
45
46          return sum;
47      }
48
49      public int[] getIDs() {
50          return ids;
51      }
52
53      public void addID(int id) {
54          int[] tmp = new int[ids.length + 1];
55          tmp[0] = id;
56          System.arraycopy(ids, 0, tmp, 1, ids.length);
57          ids = tmp;
58          Arrays.sort(ids);
59      }
60
61      @Override
62      public boolean compareID(int idx) {
63          for(int id : ids) {
64              if (id == idx)
65                  return true;
66          }
67          return false;
68      }
69
70
71      public String toString() {
72          return tsString() + " Zone event " + Arrays.toString(ids);
73      }
74
75      public boolean equals(Object o) {
76          if (!super.equals(o))
77              return false;
78
79          if (!(o instanceof ZoneEvent))
80              return false;
81
82          ZoneEvent e = (ZoneEvent) o;
83          if (e.ids.length != this.ids.length)
84              return false;
85
86          for (int i = 0; i < e.ids.length; i++) {
87              if (e.ids[i] != this.ids[i])
88                  return false;
89          }
90          return true;
91      }
```

```
92
93      /**
94       *
95       * @param id
96       * @return
97       */
98      public static List<Integer> getIDs (int id) {
99          LinkedList<Integer> ids = new LinkedList<Integer>();
100         while(id > 0) {
101             ids.addFirst(id % 256);
102             id /= 256;
103         }
104
105         return ids;
106     }
107
108     public static String getIDString(int id) {
109         if (id < 256)
110             return Integer.toString(id);
111
112         StringBuffer sb = new StringBuffer();
113         for (int i : getIDs(id))
114             sb.append(i + ",");
115
116         return sb.substring(0, sb.length()-1);
117     }
118 }
```

Listing 9: ZoneEvent.java

.1.3.5 SwitchEvent.java

```
1  package events;
2
3  public class SwitchEvent extends Event {
4
5      protected boolean cmd;
6
7      public SwitchEvent(int id, long ts, boolean cmd) {
8          super(id, ts);
9          this.cmd = cmd;
10     }
11
12     public SwitchEvent(int id, boolean cmd) {
13         super(id);
14         this.cmd = cmd;
15     }
16
17     public boolean getCmd() {
18         return cmd;
19     }
20
21     public String toString() {
22         return tsString() + " Switch event " + this.id +
23                 ((cmd) ? " on" : " off");
24     }
25
26     public boolean equals(Object o) {
27         if (!super.equals(o))
28             return false;
29
```

```
30          if (!(o instanceof SwitchEvent))
31              return false;
32
33          SwitchEvent e = (SwitchEvent) o;
34          if (e.cmd != this.cmd)
35              return false;
36
37          return true;
38      }
39  }
```

Listing 10: SwitchEvent.java

.1.4 Package: config

.1.4.1 Config.java

```
1  package config;
2  import java.io.*;
3  import java.util.Scanner;
4
5
6  public class Config{
7      public static int patternLength = 7;
8      public static int patternInterval = 10*1000;
9      public static int zoneInterval = 500;
10     public static int correlationInterval = 7*1000;
11     public static float probabilityThreshold = .5f;
12     public static boolean useZones = true;
13     public static int defaultOnTime = 5000;
14     public static int punishmentTimeout = 10*1000;
15     public static float correlationCorrectionStep = .1f;
16     public static boolean debug = true;
17     //hej David
18
19     public static void main(String[] args) {
20         Config.loadConfig();
21     }
22
23     public static void loadConfig(){
24         System.out.println("Loading Configurations");
25         try{
26             File f = new File("kiiib.settings");
27             if(!f.exists()){
28                 System.out.println("could not find preferences file,
                       generating a new one");
29                 f.createNewFile();
30                 FileWriter fstream = new FileWriter(f);
31                 BufferedWriter out = new BufferedWriter(fstream);
32                 out.write("#automatically generated preferences file\n#
                       delete to return to default settings\n");
33                 out.write("pattern_interval "+patternInterval+"\n");
34                 out.write("pattern_length "+patternLength+"\n");
35                 out.write("probability_threshold "+probabilityThreshold+
                       "\n");
36                 out.write("use_zones "+useZones+"\n");
37                 out.write("zone_interval "+zoneInterval+"\n");
38                 out.write("correlation_interval " + correlationInterval+
                       "\n");
```

```
39                  out.close();
40
41              }
42              else{
43                  Scanner scan = new Scanner(f);
44                  String token;
45                  while(scan.hasNextLine()){
46                      token = scan.next();
47                      if(token.equals("pattern_length")){
48                          patternLength = Integer.parseInt(scan.next());
49                          scan.nextLine();
50                          System.out.println("pattern_length = "+
                                patternLength);
51                      }
52                      else if(token.equals("pattern_interval")){
53                          patternInterval = Integer.parseInt(scan.next());
54                          scan.nextLine();
55                          System.out.println("pattern_interval = "+
                                patternInterval);
56                      }
57                      else if(token.equals("probability_threshold")){
58                          probabilityThreshold = Float.parseFloat(scan.
                                next());
59                          scan.nextLine();
60                          System.out.println("probablility_threshold = "+
                                probabilityThreshold);
61                      }
62                      else if(token.equals("use_zones")){
63                          useZones = Boolean.parseBoolean(scan.next());
64                          scan.nextLine();
65                          System.out.println("use_zones = "+useZones);
66                      }
67                      else if(token.equals("zone_interval")){
68                          zoneInterval = Integer.parseInt(scan.next());
69                          scan.nextLine();
70                          System.out.println("zone_interval = "+
                                zoneInterval);
71                      }
72                      else {
73                          scan.nextLine();
74                      }
75
76                  }
77              }
78          }
79          catch(IOException e){
80              e.printStackTrace();
81          }
82          catch(Exception e){
83              System.out.println("could not read preferences file... using
                    default settings");
84          }
85      }
86      }
```

Listing 11: Config.java

.1.5 Package: core

.1.5.1 Correlation.java

```
1  package core;
2
3  import java.io.IOException;
4  import java.sql.Connection;
5  import java.sql.DriverManager;
6  import java.sql.ResultSet;
7  import java.sql.SQLException;
8  import java.sql.Statement;
9  import java.util.Arrays;
10 import java.util.HashMap;
11 import java.util.HashSet;
12 import java.util.LinkedList;
13 import java.util.List;
14 import java.util.Map;
15 import java.util.Set;
16 import java.util.TreeSet;
17
18 import timer.TimeoutEvent;
19 import timer.TimeoutListener;
20
21 import config.Config;
22
23 import events.*;
24
25
26 public class Correlation implements TimeoutListener {
27
28     private Statement stmt;
29     private Connection conn;
30     private ResultSet result;
31     private long correlation_interval = 7*1000;
32     private float correction;
33     private Map<Integer, Map<Integer, Float>> correlation;
34
35     public static void main(String[] args) throws IOException {
36         System.out.println(System.currentTimeMillis());
37         System.in.read();
38     }
39
40     public Correlation () {
41         correlation = new HashMap<Integer, Map<Integer, Float>>();
42         try {
43             Class.forName("com.mysql.jdbc.Driver");//load the mysql
                   driver
44             conn = DriverManager.getConnection("jdbc:mysql://localhost/
                   kiiib?user=KIIIB&password=42");//connect to the database
45             stmt = conn.createStatement();
46         }
47         catch (SQLException se){
48             System.out.println("SQLException: " + se.getMessage());
49             System.out.println("SQLState: " + se.getSQLState());
50             System.out.println("VendorError: " + se.getErrorCode());
51
52         }
53         catch (Exception e){
54              e.printStackTrace();
55         }
56         correction = Config.correlationCorrectionStep;
57         generateCorrelation();
```

```
58 //          getStoredCorrelations();
59     }
60
61     public float getCorrelation(int switchId, int sensorId) {
62         if (! correlation.containsKey(switchId))
63             return 0;
64
65         if (! correlation.get(switchId).containsKey(sensorId))
66             return 0;
67
68         return correlation.get(switchId).get(sensorId);
69     }
70
71     public static void incrementSwitchCount(Map<Integer, Integer>
           switch_count, int id) {
72         if (!switch_count.containsKey(id))
73             switch_count.put(id, 1);
74         else
75             switch_count.put(id, switch_count.get(id) + 1);
76     }
77
78     public static void incrementSensorCount(Map<Integer, Map<Integer,
           Integer>> sensor_count, int switchId, int sensorId) {
79         if (!sensor_count.containsKey(switchId)) {
80             sensor_count.put(switchId, new HashMap<Integer, Integer>());
81         }
82
83         Map<Integer, Integer> map = sensor_count.get(switchId);
84         if (!map.containsKey(sensorId)) {
85             map.put(sensorId, 1);
86         } else {
87             map.put(sensorId, map.get(sensorId) + 1);
88         }
89     }
90
91     private void updateCorrelation(int sw, int se, float corr) {
92         if (correlation.containsKey(sw)) {
93             Map<Integer, Float> map = correlation.get(sw);
94             if (map.containsKey(se)) {
95                 map.put(se, Math.max(0, map.get(se) + corr));
96             }
97         }
98     }
99
100    public void generateCorrelation() {
101
102        try {
103            Map<SwitchEvent, EventList> switch_eventlist = new HashMap<
                   SwitchEvent, EventList>();
104            Map<Integer, Integer> switch_count = new HashMap<Integer,
                   Integer>();
105            Map<Integer, Map<Integer, Integer>> sensor_count = new
                   HashMap<Integer, Map<Integer, Integer>>();
106            LinkedList<SwitchEvent> gc = new LinkedList<SwitchEvent>();
107
108            result = stmt.executeQuery("(select id,timestamp,'sensor' AS
                    type, '0' AS status from sensor_events) union (select
                   id,timestamp,'switch' AS type,status from switch_events)
                    order by timestamp;");
109            int i = 0;
110            while(result.next()) {
111                int id = result.getInt("id");
112                long ts = result.getTimestamp("timestamp").getTime();
113                if (result.getString("type").equals("switch")) {
```

```
114                     boolean cmd = (result.getInt("status") == 1) ? true
                            : false;
115                     if (cmd) {
116                         SwitchEvent s = new SwitchEvent(id, ts, cmd);
117                         switch_eventlist.put(s, new EventList(Config.
                                zoneInterval, Config.correlationInterval,
                                Integer.MAX_VALUE));
118                         gc.addLast(s);
119                     }
120                 } else if (result.getString("type").equals("sensor")) {
121                     for (SwitchEvent e : switch_eventlist.keySet()) {
122                         if (e.getTS() + correlation_interval > ts) {
123                             switch_eventlist.get(e).add(new SensorEvent(
                                    id, ts));
124                         }
125                     }
126                 }
127
128                 while(gc.size() > 0 && gc.getFirst().getTS() +
                        correlation_interval < ts) {
129                     SwitchEvent se = gc.getFirst();
130                     incrementSwitchCount(switch_count, se.getID());
131
132                     for (Event e : new HashSet<Event>(Arrays.asList(
                            switch_eventlist.get(se).getEvents()))) {
133                         incrementSensorCount(sensor_count, se.getID(), e
                                .getID());
134                     }
135                     gc.removeFirst();
136                     switch_eventlist.remove(se);
137                 }
138
139                 for(int sw : sensor_count.keySet()) {
140                     Map<Integer, Float> map = new HashMap<Integer, Float
                            >();
141                     for (int se : sensor_count.get(sw).keySet()) {
142                         map.put(se, (float) sensor_count.get(sw).get(se)
                                / switch_count.get(sw));
143                     }
144                     correlation.put(sw, map);
145                 }
146             }
147             while(gc.size() > 0) {
148                 SwitchEvent se = gc.getFirst();
149                 incrementSwitchCount(switch_count, se.getID());
150 //                for (int sensor : switch_events.get(se))
151 //                    incrementSensorCount(sensor_count, se.getID(),
    sensor);
152                 for (Event e : new HashSet<Event>(Arrays.asList(
                        switch_eventlist.get(se).getEvents()))) {
153                     incrementSensorCount(sensor_count, se.getID(), e.
                            getID());
154                 }
155                 gc.removeFirst();
156 //                switch_events.remove(se);
157                 switch_eventlist.remove(se);
158             }
159         } catch (SQLException se){
160             se.printStackTrace();
161             System.out.println("SQLException: " + se.getMessage());
162             System.out.println("SQLState: " + se.getSQLState());
163             System.out.println("VendorError: " + se.getErrorCode());
164         }
165     }
166
```

```
167     public Set<Integer> getSwitches() {
168         return new TreeSet<Integer>(correlation.keySet());
169     }
170 
171     public Set<Integer> getSensors() {
172         Set<Integer> sensors = new TreeSet<Integer>();
173         for(int sw : correlation.keySet()) {
174             sensors.addAll(correlation.get(sw).keySet());
175         }
176         return sensors;
177     }
178 
179     /**
180      * get a list of switches, that have a correlation with a sensor
           above the threshold
181      * @param sensor
182      * @param threshold 0 <= x <= 1
183      * @return
184      */
185     public List<Integer> getSwitches(int sensor, float threshold) {
186         List<Integer> list = new LinkedList<Integer>();
187         for (int sw : correlation.keySet()) {
188             Map<Integer, Float> map = correlation.get(sw);
189             if (!map.containsKey(sensor))
190                 continue;
191 
192             if (map.get(sensor) > threshold)
193                 list.add(sw);
194         }
195         return list;
196     }
197 
198     public String toString() {
199         StringBuilder sb = new StringBuilder(1024);
200         sb.append("Corr.\t");
201         for (int s : getSensors())
202             sb.append(ZoneEvent.getIDString(s) + "\t");
203         sb.append("\n");
204 
205         for (int sw : getSwitches()) {
206             sb.append(sw + "\t");
207             for (int se : getSensors()) {
208                 if (correlation.get(sw).containsKey(se)) {
209                     float f = correlation.get(sw).get(se);
210                     if (f >= 0.5)
211                         sb.append("*");
212                     if (f > 0)
213                         sb.append(String.format("%.2f\t", f));
214                     else
215                         sb.append("\t");
216                 } else {
217                     sb.append("0\t");
218                 }
219             }
220             sb.append("\n");
221         }
222         return sb.toString();
223     }
224 
225     @Override
226     public void TimeoutEventOccurred(TimeoutEvent event) {
227         // TODO Auto-generated method stub
228 
229     }
230 
```

```
231     public void increaseCorrelation(int sw, int se) {
232         System.out.println("Increase correlation " + sw + "~" +se);
233         storeCorrelation(sw, se, Config.correlationCorrectionStep);
234         updateCorrelation(sw, se, correction);
235         storeCorrelation(sw, se, correction);
236     }
237
238     public void reduceCorrelation(int sw, int se) {
239         System.out.println("Reduce correlation " + sw + "~" +se);
240         storeCorrelation(sw, se, -Config.correlationCorrectionStep);
241         updateCorrelation(sw, se, -correction);
242         storeCorrelation(sw, se, -correction);
243     }
244
245     public void getStoredCorrelations() {
246         String query = "SELECT switch, sensor, correlation FROM
                correlation_confirmation";
247         try {
248             result = stmt.executeQuery(query);
249             while(result.next()) {
250                 int sw = result.getInt("switch");
251                 int se = result.getInt("sensor");
252                 float corr = result.getFloat("correlation");
253                 updateCorrelation(sw, se, corr);
254             }
255         } catch (SQLException ex){
256             ex.printStackTrace();
257             System.out.println("SQLException: " + ex.getMessage());
258             System.out.println("SQLState: " + ex.getSQLState());
259             System.out.println("VendorError: " + ex.getErrorCode());
260         }
261     }
262
263     /**
264      * insert correlation correction into sql table
265      * @param sw switch id
266      * @param se sensor id
267      * @param corr correlation change
268      */
269     public void storeCorrelation(int sw, int se, float corr) {
270         String query = String.format("INSERT INTO
                correlation_confirmation " +
271             "(switch, sensor, correlation) VALUES (%d, %d, %f) " +
272             "ON DUPLICATE KEY UPDATE correlation = correlation + %f; ",
                    sw, se, corr, corr);
273         try {
274             stmt.executeUpdate(query);
275         } catch (SQLException ex){
276             ex.printStackTrace();
277             System.out.println("SQLException: " + ex.getMessage());
278             System.out.println("SQLState: " + ex.getSQLState());
279             System.out.println("VendorError: " + ex.getErrorCode());
280         }
281
282     }
283 }
```

Listing 12: Correlation.java

.1.5.2 KeyList.java

```
1  package core;
2  import java.util.ArrayList;
3  import events.*;
4  /**
5   * class keylist
6   * */
7  public class KeyList{
8      private ArrayList<Integer> keys;
9      public KeyList(){
10         keys = new ArrayList<Integer>();
11     }
12     public KeyList(EventList elist){
13         keys = new ArrayList<Integer>();
14         for(Event e : elist.getPattern()){
15             keys.add(e.getID());
16         }
17     }
18     public int hashCode() {
19         int hashcode=0;
20        for(int i : keys){
21             hashcode = hashcode*31 +i;
22        }
23        return hashcode;
24     }
25     public boolean equals(Object o) {
26         try{
27             KeyList a = (KeyList)o;
28             if(this.size() != a.size()){
29                 return false;
30             }
31                 for(int i=0;i<keys.size();i++){
32                     if(this.get(i)!=a.get(i)){
33                         return false;
34                     }
35                 }
36                 return true;
37         }
38         catch(Exception e){
39             return false;
40         }
41     }
42     public void add(int i){
43         keys.add(i);
44     }
45     public void add(int k, int i){
46         keys.add(k,i);
47     }
48     public int get(int k){
49         return keys.get(k);
50     }
51     public int size(){
52         return keys.size();
53     }
54     public KeyList subList(int x, int y){
55         KeyList k = new KeyList();
56         for (int i = x;i<=y;i++){
57             k.add(keys.get(i));
58         }
59         return k;
60     }
61     public void printValues(){
62         for (int i : keys){
63             System.out.print(i+" ");
64         }
```

```
65      }
66      public ArrayList<Integer> getKeys(){
67          return keys;
68      }
69    public String toString(){
70      String returnstr = "";
71      for (int i : keys){
72        returnstr = returnstr+i+" ";
73      }
74      return returnstr;
75    }
76  }
```

Listing 13: KeyList.java