A Distributed, Collaborative Development Project of a Policy Engine for Building Automation

Global Software Development

Berntsen, R., raber@itu.dk
Hansen, K., kben@itu.dk
Kokholm, T., tkok@itu.dk
Stanciulescu, S., scas@itu.dk
Wainach, N., nicl@itu.dk

May 14, 2013

Abstract

This paper presents a distributed, collaborative software prototype development project in the domain of building automation. The prototype system — the policy engine – is used for regulating the internal environment of a building. The global team behind the prototype consists of members from Strathmore University, Kenya and the IT-University of Copenhagen, Denmark.

The field of building automation has seen a lot of growth in recent years. Building automation can save energy and resources by e.g. turning off light or regulating the temperature when nobody is in the room. Conservation of energy and natural resources through building automation offers wast improvements compared to buildings where this automation is not deployed.

The approach to the project is to analyse, design and implement the policy engine prototype in a distributed, collaborative project. First we created the communication platform for the project. Then we managed the project while simultaneously developing the prototype. During the length of the project we founded multiple subgroups in regards to team members preferences and interests - seeking to optimize both our quality and quantity of code.

The end result was a fully working prototype of a policy engine platform, offering functionality to create, edit and execute policies. The engine was validated through a series of tests, including low level functionality tests — unit tests — and high level user tests.

We argue that our approach towards the development of the prototype is both valid and appropriate in relation to the goals, requirements and audience.

Contents

Chapter 1

Introduction

The research field of building- and home automation experiences a lot of growth – not least because of the promise to reduce energy consumption by more intelligent control, but also due to the possibility of heightened human comfort. Constructing resource efficient buildings makes sense, both in a political and economical perspective. In [14] it is stated that residential buildings use about 82% in total energy consumption on space and water heating. Electric appliances uses 11%. One will be able to save on these sources by using policies. Policies can be seen as a general rule that determines what to do in sense of controlling lights, heat, blinds etc. An example of a policy could be turning off heating at midnight and turn of lights in room where there is nobody present. Manually controlling an energy-saving policy is very time-consuming, inefficient task and the user has to know a lot of information on different equipment. As well as information about the building itself. The process is however achievable, using building automation.

Regulating the environment of buildings to heighten human comfort, requires that building components can communicate and be controlled in a fashion that seems intuitive for human beings. Buildings today might come equipped with a suite of sensors and actuators, opening up for a degree of customizable control. Our collective need is that buildings can adapt to the users and the sensor-perceived environment. We use the term policy to describe a building automation program, and the term policy engine as the overall software entity that controls the aforementioned policies.

Trivial policies can be based on semi-static data, like time and weekdays. However this can have unforeseen and unwanted consequences. For example, a policy governing lightning activated merely by a static time schedule, might entail problems for people attending a rarely occurring late-night party in the building. If

the event calendar of the building is accessible to the policy engine, a conditional event-checking statement might ensure continuous lighting. However, in order to achieve a more fine grained control, sensory input is needed. The interaction with these policies are thought of and developed in respect of a facility management[1]. The task of controlling the policies is also defined as being part of the tasks of FM.

By employing a policy engine, with access to the buildings sensors and actuators, both the building owner, the users and the administrators benefit from the automation provided. If policies are efficiently defined, building owners could save energy and other natural resources while still providing extra comfort to their tenants. Users will experience a building that autonomously adjust its internal environment to suit the comfort and needs of its users. While FM administrators will achieve a more fine-grained control of the building.

As stated in [7], efficient energy saving is only possibly with building automation, known as "BA": *Worldwide, there is no doubt that efficient energy saving is only possible with modern BA based on networking in all levels of abstraction.*

This project was defined in the course Global Software Development at IT University of Copenhagen[2]. Our global development team consists of 5 students from ITU and 4 from Strathmore University in Nairobi, Kenya. The team is provided with a Building Simulator, making up for the lack of a real constructional building. The complexity with integrating hardware and communication protocols are thus avoided. However the simulator has provided other complexities which will be evident later in this paper.

Throughout the development of this project the focus has been geared towards the simulator. The end product is a web-based management application, that allows for centralized flow, and control between sensors and actuators in the simulated building. This is achieved by implementing a policy engine that allows for automated actuator-responses based on sensor feedback. An example of this could be closing the blinds in excess sunlight or turning off the heaters while windows are open.

In this paper we will:

1. Distil requirements from course provided material.
2. Literature (re-)search on policy engines.
3. Develop a software solution that implements these requirements.

[1] Hereby denoted: FM
[2] Hereby denoted: ITU

4. Document the collaborative project between the IT University of Copenhagen, Denmark and Strathmore University, Nairobi Kenya.

1.1 Context

Modern buildings get more and more complex – from the type of materials being used, to the services and infrastructure they provide. Our focus in this paper is building automation through the use of governing policies, and therefore economically speaking, relates to the buildings running costs, without lack of focus on human comfort. Today the cost of resources such as gas, diesel and electricity have been generally climbing for the last many years, making it still more important to use them with care. Another cost related to the usage of energy and natural resources are the taxes which also are on the rise. However, economy is not the only factor for saving the planets resources. Today it is considered so good PR to 'go green' that some companies build a virtual presence for this subject alone [11] [9] [28].

A part of the whole 'go green' concept is to control your energy and natural resource usage. Many companies have buildings that are either heated or cooled (or both), have lighting, appliances and server rooms to name a few. They may already be equipped intelligent heating systems, HVAC systems[3] and AC's[4] but many of these systems are typically proprietary and impossible —or very hard—to integrate with. As mentioned earlier this project integrates into a Building Simulator. This makes our task more simple compared to real work applications. We do not have to worry about actual sensor- and actuator hardware, wiring, costs, communication protocols, existing protocols, coverage of wireless communication signals and so forth.

We assume that the Building Simulator *is* the building. Our system is therefore *open* compared to proprietary, since all it takes to access (e.g. the temperature in a room) is a http get operation with the building id and sensor in question. A everyday example could be a janitor, who might need to go to a room physically to check its temperature, and adjust the heater in that room, several times a day to achieve and maintain the desired temperature. Our governing policies makes it up for a building janitor, or at least the trivial work with adjusting room temperatures and the likes. Human staff starts out by defining the policies that the building needs. Then we can imagine the virtual janitor, issuing http get operations all day, to check the sensors mentioned in those policies. The policies consists of sensor-conditional behaviour that will adjust the actuators continuously. This is done by issuing http post commands to the building simulator with

[3] Heating, Ventilation, and Air Conditioning
[4] Air Conditioning

sensor id and the desired value.

1.2 Problem

With the ability to manage every sensor in a given building, one can assume that the task of controlling these gets progressively —for each number of additional sensor and actuator—more complex. Any additional concern will add further complexity, such as optimizing the control to prioritize both economic and human comfort. One needs to consider this challenge: What good is a highly advanced building if the task of controlling it is just as complex?

We seek to design a system where FM can define a set of policies they believe is needed, based on their experience with the building. We do not expect FM to be IT experts, but we do expect them to be able to use a modern browser and be familiar with the sensor id's used in the building.

Our main consideration in this paper is: *Providing a web-based infrastructure to visually define, manage and monitor the scheduling of governing policies for a building. The policies should be executed at regular intervals, based on their time specific settings. The infrastructure should be provided in an easy accessible interface that provides FM with the means of controlling the building's actuator systems, based on sensory input and conditions expressed in the policies. It should be possible to define policies that will maintain human comfort in that building, while conserving energy with regards to heating and/or lighting.*

In addition to the general problem from above, we further want to point out these challenges:

1. The rating of proposed solution's usability.
2. Policy time control and manual override.
3. The use of complex policy data structures on both frontend and backend.

1.3 Learning Goals

The project in itself consists of two widely different aspects. They are the global software development and the policy engine. The new aspect in the project life cycle is for us how to collaborate with team members distributed in different continents and countries, with many differences and thus challenges to overcome.

These two aspects are also the ones the main learning goals are derived from. In this regard the goals can be grouped into two different parts. The first being how to collaborate across different countries and the challenges that this potentially could cause, including how to manage the team, how to communicate, how to increase performance and the likes. The secondary being how we can solve the technical project and deliver a working prototype of a policy engine, including creating the domain model, data structure, creating the actual policy engine in a object oriented-manner and tying this together with the visual and user friendly frontend.

We could summarize the learning goals to be:

- How to work together in a distributed team and reflect on this.
- How to develop a policy engine that complies with the stated requirements (See section 1.4).

1.4 Requirements

The requirements of the project are stated in a very open-ended manner, with only a few descriptive requirements to the product and process. These are formulated as requirements to the product and process. They are:

Product

- The students analyse their solution from their chosen sensors and actuators requirements (and additional requirements they can think of) into functionality for their application (30%).
- The students must design and implement a Web/Android application to monitor, control and visualize policies in a building (20%).
- The students must describe and evaluate their solution as used by facility people as metrics. Additional metrics can be considered and will be taking into consideration by the examiners (20%).

Process

- The students should be able to write a proper, understandable and well organized report (10%).
- The students should be able to reflect on a real world collaborative experience (20%).

Chapter 2

Related Work

A lot of research has emerged lately in the field of energy management systems for smart buildings. A similar work with the one presented in this paper has been done by Tiberkak et. al [32], where a Policy Based Architecture for Home Automation System is developed. The system is composed of the following subsystems: one responsible for home automation, one for the local management of rooms, one for storing data, and a system for enabling communication between the first two sub systems. Different profiles are defined to improve energy efficiency. The concept of preferred and required profile is introduced, to differentiate between the preferences of a inhabitant and the maximum and minimum values of each environmental parameter in the required profile. Notifications and messages are sent between the layers when there is a change in the status of a room, and a appropriate decision is taken. Their approach is focused more on the comfort of a person in the house than our approach, which is focused more on the management of a bigger building with more sensors and actuators. This approach could be used if a room in a building is known to be used frequently by the same persons, so one could define its own profile or policy.

Another approach is the one taken by Li et. al [21] where they implement a energy management system for homes, that provides automatic metering and capability of taking decisions based on monitoring energy consumption. Tasks can be used to specify different actions required at different moments. A simulation has been done for 1000 homes and by using their system, they achieved a significant energy reduction.

In [12] a complete system for home management is described. Using ZigBee and IEEE 802.15.4, it is easy to add new devices, connect them with already existing ones and control them using a remote device. However, no kind of automatic management for the light, heating and air conditioning services exists. This

research would be useful for our work if there would be the need to add more sensors or actuators to the building.

In [34] the authors develop a prototyping platform based on Building Controls Virtual Test Bed framework [30] for controlling and testing networked sensors and actuators on physical system. An algorithm for controlling the light and blind system in the room has been developed. The system is configured to provide an illumination of 500 lux between 6 AM and 6 PM. This has been achieved by measuring the daylight and setting the blind to block direct sun beam into the room. The system managed to achieve a reduction by an average of 17% of cooling demands and a maximum of 57% of lighting demand compared to the reference room. Their tests have been conducted on actual rooms, which is not possible in our case. This has been optimized to provide the optimum illumination in a room but in furthher development of our work we could use this research to include default policies that would control the illumination of a room, in order to maximize the energy reduction.

Krioukov et al [18] take another approach with their Building Application Stack. They provide an application programming interface and runtime for portable building applications. Developers can express their intent in a easy way e.g. " turn off the lights for top floor cubicles near windows" or can handle building differences by having support for programatic exploration of the building's components. It consists of three layers, the query based API layer, the driver layer and the abstract interconnection layer. The query layer provides functionality for selecting objects based on attributes, type and functional or spatial relationships. The driver layer is composed of high-level and low-level drivers to encompass the number of different components in a given building. The low-level defines the types of the components and the high-level provides descriptive information of those components. The abstract interconnection layer is a RESTful interface to facilitate communication between the application's request and the building control protocols. They use tags to describe the types of objects, for example #FLOOR or #LIGHT for selecting all the objects on all floors and all objects in lighting domain respectively. This led us to create wildcards in our system, which work similar to their tags, but are more specific.

Chapter 3

Collaboration Strategy

In this chapter we present our tactics and methods in regards to the collaboration. The level of abstraction is primarily on a descriptive level, while some challenges are also reflected upon.

3.1 Workflow

For this project we decided to use an iterative design approach, both for the front-end and the back-end. Using this approach we were able to continuously evaluate both the concept and software.

We also used informal testing throughout the entire project. The goal here was to continuously test and optimize the software, learn about the stability and performance issues, so to finally adjust the implementation according to these findings. An iterative design approach is commonly used for front-end design. It allows designers to identify usability issues that may arise in the user interface - before it is put into wide use. Even the best usability experts[1] cannot design perfect user interfaces in one single attempt. Hence a usability engineering life cycle should be built around the concept of iterations [23].

Figure 3.1 illustrates the different iterations, phases and stages of our project.

Each of these illustrated elements were important for the process and contains all valuable information. Figure 3.1 contains a timeline, which chronologically shows, how the project evolved. The project went through three overall stages. These are known as the "Initial Stage", "Development Stage" and finally a "Closing Stage".

[1] We recognize Jakob Nielsen as a widely known usability expert.

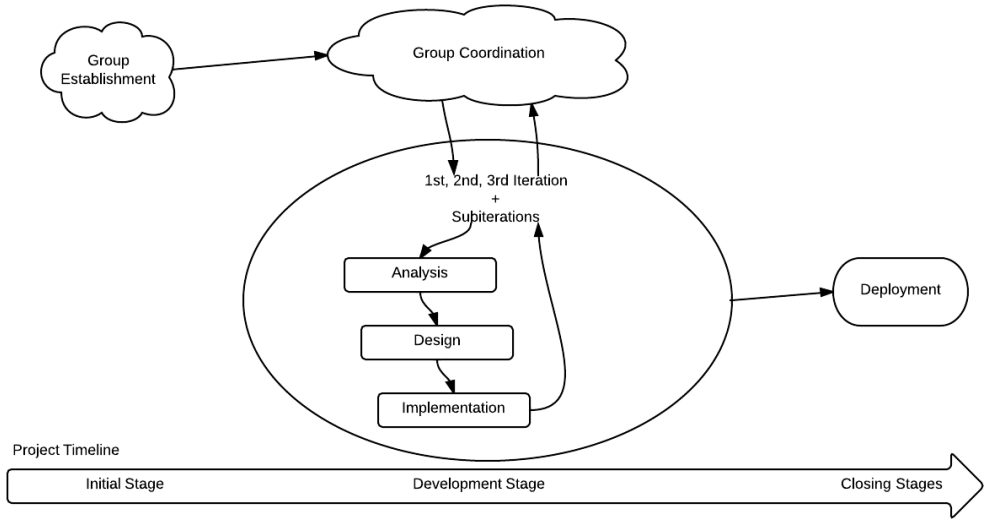

Figure 3.1: A generalized view of our workflow process

3.1.1 Stages

3.1.1.1 Initial Stage

Before the project kicked off and any kind of implementations began, we focused on how to handle the nine people in the group as *one* collaborating team. This part was of high importance to us, being students with different nationalities, from far different continents. We didn't know anything about each other in advance. This made the task quite complicated. One could presume that in a normal work setting, at least the different roles in a team would already be settled. One could imagine that some people were hired to do the back-end (developer) and other people in charge of managing the project (project manager). We needed a specific initial stage to cover this part of getting to know each other, figuring out roles and knowledge areas of each team member. Within the group this was known as the "Initial Stage". The stage established the necessary basis to work as a team and made sure that project development went in the right direction to begin with.

3.1.1.2 Development Stage

During the development stage several iterations of analysis, design and implementation were executed. As figure 3.1 shows: The repetitive phases in the development stage is an iterative process. We went through this loop multiple times. But to summarize we had three major iterations, as described in section

3.1.2.

The iterations are coherent with the project plan and present our general approach to the project. All of the three iterations contains multiple sub-iterations. Describing each iteration in detail would be a somewhat cumbersome process.

The reason for choosing three major iterations was to accommodate the overall deadlines for this project. Each deadline requested different material containing information about different levels of the project. The first deadline was focused on the requirements and motivation. The second delivery focused on related work and a project model, while the third delivery focused on the source code. These deadlines were a part of our overall project plan and thus led to having three overall deadlines. Initially our goal was to have four major iterations but this was changed due to our unsuccessful efforts to include our colleagues from Kenya. The postponement of deadlines and a general patient attitude caused us to eliminate one major iteration.

The development stage contains furthermore a group coordination phase. The coordination is reached multiple times throughout the iterations. During this phase important topics were covered, such as sharing status, separation and distribution of tasks, figure out how to proceed, discuss upcoming challenges and the likes. Multiple collaboration methods and tools were used throughout this coordination phase (See 3.2 and 3.2.2).

3.1.1.3 Deployment Stage

The deployment stage was considered the simplest and easiest to handle stage of our process. Upon reaching agreement that the development was finalized, the product was deployed to our server. This concluded our work on the product and finalized the development.

3.1.2 Iterations

3.1.2.1 First Iteration

Our focus during the first iteration was to get the group work to flow, and figure out how to handle the data services provided by the Danish university. Most of the iteration was related to get to know each other and figure out how to move the project forward. The artefacts produced here was of lower level, such as diagrams of the group members programming skills, use case diagrams, first thoughts of domain model, early prototypes of the front-end interface and early implementations of the data services.

It was difficult to push the project forward at this phase. We used different means of communication with the Kenyans, but without any positive result. Our best result was reached when two out of the four Kenyans members replied to our emails. No feedback, replies, comments or the likes was received from the two last members.

3.1.2.2 Second Iteration

During the second major phase the groups communication platform and general condition was stable - but unfortunately without any involvement from our Kenyan team members. We knew that we had to push the project forward, and had already wasted too much time on the effort to engage and activate unresponsive members. We got way behind our project plan in an attempt to give them enough time to respond, without any luck. We tried to activate one semi-active member from Kenya as a team leader, but his response was, that he was not able to contact the other members.

Our focus changed from here. At the following team meeting we agreed upon, that it would be up to us, to get the project going with or without the Kenyan team members. We chose to do so due to the fact that we did not receive any kind of feedback from Kenya. We would still let them know every step in the project, and we would continue to try to involve them as much as possible - by emailing them and the likes. We developed multiple artefacts at this point. Our goal through the second iteration was to get a working backend up and running. To do this, we had to chose the best fitting development platform and hereby programming language. The group had to agree on a set of requirements for the engine derived from the requirements in the assignment. The data model was completed and implemented, development of an early working prototype of the backend, and generated several jUnit tests to validate its robustness. Furthermore the work on the frontend started to take form. We did our first evaluation of the frontend (see 6.2).

3.1.2.3 Third Iteration

The third and final iteration was a hectic period with focus on getting things done and complete tasks. Because of the earlier effort to activate the Kenyan members, and the time consumed here, we were lacking behind. The now only moderate active member from Kenya chose to leave the group due to personal circumstances. The frontend was implemented with support from data provided by the backend. A second usability test was produced. Also this paper was mostly produced here, by combining bits and pieces and adding lots of new material.

Finally the end product was deployed to our server.

During all of the iterations, both major and minor, we used a palette of different methods and tools. These are highly linked with the project phases. The most important are introduced and explained by usage, in the parts that follow below.

3.2 Collaboration

A set of different methods were used during the initial stage, in the group establishment, and afterwards in the development stage. These were applied in order to promote and streamline collaboration. Several challenges were met and could have been solved by applying the correct mitigation strategy. The collaboration between the Danish and Kenyan team never evolved enough to engage some of the tactics. The following sections will only highlight the actual engaged methods, and will leave also leave out some methods that are only used briefly. This is done to inform which important actions were taken to engage the Kenyan members and not to blur the picture by which methods that could have been used at a later stage, if the collaboration evolved.

3.2.1 Create a common platform

At the time of the project start we were fully aware that we had several discontinuities, as described by Watson-Manheim et al. [33], between the two groups. Too mention a few, the most challenging were, that we did not share the same culture, time zone —Kenya was ahead of two hours—, possibly skills, history and the likes. All of which makes it a challenge to collaborate. One way to solve this is to get to know the different team members in the group. The first step we took in order to make the group work as one unit, was to try and create a common ground. The term to create common ground "refers to that knowledge that the participants have in common, and they are aware that they have it in common" [26]. The fact that the members in the teams, local as well as global, had not met each other, or collaborated, before, only stressed the importance of this. In a group like this, where members are spread across different continents, the common ground and the social context that one behave in, is important. Nobody knew what to expect from each other, and different cultures introduces different challenges.

Our approach to mitigate this problem was to email each team member, using the emails supplied by the coordinators, and introduce the team members from ITU. The introduction was superficial and not exhaustive, in order not to frighten anybody. Basic information was listed, such as name, gender, age, contact meth-

ods, and the likes. This would be a 30 seconds email to compose. However we never received any replies. Despite several tries. To mitigate this challenge of having no replies, we tried to create an online spreadsheet, listing these simple informations, and expanded it with individual skills that were significant to this project. It took the Kenyan members close to two weeks to complete this contact form, and some Kenyan team member never completed it.

The reasoning for collecting the members individual skills was to figure out which platform we could develop on. Using a development platform from which only few team members have knowledge about wouldn't be ideal. Hence it was crucial to obtain this information before deciding.

In the beginning of the project we talked about the many technologies available for web development. There are many approaches to the same solution, all with their pros and cons. Focusing on working as a team, with different members that all have considerably different background experiences - we decided to individually write down our development qualifications.

The results were collected and the top three choices were joined in a table (See Table 3.1). Every member then voted for a preferred development platform. One (1) being the preferred approach and three (3) the least preferred. The results are shown below:

Table 3.1: Members preferred development platform

	Kasper	Thomas	Stefan	Rasmus	Nicolas	Steven	Cecil	Tom	Lucy
PHP	2	1	3	3	1	1	3	-	1
C#.Net	3	3	2	1	3	3	2	-	3
Java	1	2	1	2	2	2	1	-	2

Note: Tom in table 3.1 never got to share his experience with the rest of the team. Therefore the comparison table was never fully completed.

The table shows a total score of 20 points to C#, 15 points to PHP and finally 13 points to Java (lower is better). This indicated that Java could be the right choice for this project in regards to technology readiness.

One of the challenges in this project was indeed technology readiness [26]. Every team member has a preferred development language, which ultimately led to a long debate on settling with a development language - including what kind of framework the group should utilize. The problem was that some team members, would have some difficulties participating in the development - due to the fact that not all have the same preferred language.

Also some members already had some experience in developing web applications, while others had none, or very little. Not everyone was familiar with MVC based development, which for those unfamiliar with MVC, became a challenge. The overall problem was simply having a different level of knowledge regarding web development and technologies involved.

Ultimately, however, the group decided to go with Java as the main programming language, combined with JavaScript for the frontend. This was chosen due to the fact that Java was the most popular in the table 3.1, and that we couldn't see any limitations with this choice. Hence it seemed to fit our needs in developing a policy engine.

Various milestones for the project was planned, and the different implementation tasks, was assigned to different team members. Most noticeable the team created subgroups. The subgroups were divided in team-members working with front-end tasks and members working with back-end tasks. However frequent meetings and online conversations, led to a positive awareness [20] on what everyone was working on.

But the fact that Kenyan team members took several days, and even weeks, to reply our emails created a tense feeling in the group. The Danish group had high hopes for the collaboration and were left with a perception of sluggishness towards the Kenyan group. This caused that several members lost trust in the Kenyan members - as introduced by Jarvenpaa et al. [15]. It shows how important first impression is.

To make it easy to communicate between the group, we created an distributive email list, which every Kenyan and Danish team member was assigned to. This was made to enable members to create a global awareness in the team and to easily inform each other, which challenges we currently faced. We informed every member on how to use this email. Note that only very few emails was received from the Kenyan group, and that they were all from one particular team member. The remaining members of Kenya didn't responded. The few emails we got from the only responding Kenyan member, were often very late and inadequate.

3.2.2 Tools

In total we used six different tools to communicate about, our teamwork and progress; one synchronous and five asynchronous. We used Skype for every team meeting, and for general communication between the members of the group. Skype is considered synchronous while the communication normally is instant replies - using the voice chat. The remaining five tools we used are all considered

asynchronous communication methods: GitHub[2], Email, Google Drive[3], Doodle[4] and Team Work Project Manager[5].

GitHub was used to share the project source code. Team Work Project Manager were used to schedule and share current status of what needs to be done. Email was used for communicating general group announcements, while we wanted to make sure that all of the members from the group received the message. A total of more than 200 emails were exchanged during three months of collaboration. Google Drive to share important documents, such as general to do lists and spread sheet with member interests and skills.

In order to mitigate the time zone differences between Nairobi and Copenhagen, that could create problems, we used Doodle to try to sketch up a normal group members week (See appendix A.4). Doodle is a free online scheduling tool, that automatically takes time zones in to account. One gets presented with a scheme of days and given periods on this day, such as Monday 18th of February, 8:00 to 12:00. Then selects the hours and days that fits. This was done to sketch up the available hours and days that matched the group best. Five Danish members and one Kenyan member filled out this form. It nonetheless told us, that we would have problems meetings different days than Tuesdays and thus had to rely on the other mentioned asynchronous tools to communicate.

Finally we used Team Work Project Manager to handle deadlines and milestones, and generally to keep each other informed of the progress. It also includes a Gantt chart that every member could access (see appendix A.3).

3.2.3 Global Collaboration - with a lack of global

Even though we were only in the early stages of the team work it became quite clear to us that the collaboration was not going to be successful on a global perspective. We only managed to get limited contact with two persons from Kenya, and never managed to either skype chat (neither text or voice) or email with the last two members. One thing that could cause their lack of interaction in our e-mail correspondences and skype meetings are technology readiness. We know that it is a challenge for some of the group members to get internet access. A recent study shows that only 36,3 % have access to internet in Kenya [25], which compared to Denmark's 86 % in 2010 [8], indicates that it is not as easy to get internet access in Kenya as we are used to in Denmark. And we know by fact that a few of them do not have a stable internet connection in their home.

[2] `https://github.com/tkok`
[3] `https://drive.google.com`
[4] `http://www.doodle.com`
[5] `http://www.teamworkpm.net`

Our approach to solve this issue was to have a meeting each Tuesday at 10:00 GMT +1. This way we knew that they should have access to their University, which on normal circumstances has an internet connection they could use. We know they should have time for this meeting, thus it is planned as a course on their schedule. Secondly it might have been caused by collaboration readiness. Collaboration readiness is a potential show stopper for any team work, if a given member is not ready to collaborate. There could be many reasons for this, one being that the student was unsure if he or she was going to follow the course. We have tried to identify these issues towards our fellow group members and we cannot find anything that should indicate that they would not be willing to collaborate. They should be just as interested in delivering a good product. We never received any other indication or statement telling us otherwise.

3.2.4 Mitigation

We feel, as a group, that we were well prepared for this collaboration. We only got to use a small subset of our methods to mitigate problems. This was due to the fact that no real intentional collaboration effort was there. It is quite obvious to us that no different tactic, or set of methods, could have solved these problems. If the collaboration had moved from the initial stage, and reached an actual level of real collaboration, one could probably argue that our approach was relied on wrong methods or the likes. However this was not the case. It failed at the earliest stages, before we had any real impact. These challenges, and our thoughts on what could have been done differently, and how the problems could have been mitigated, will be discussed and reflected more upon in section **??**.

Chapter 4

Design

With inspiration from other building management systems (see Section 2) and looking at the building simulator we had to communicate with, we came up with the concept of policies that consist of two types of statements; *if-statements* and *set statements*. If-statements works as if-statements in many popular GPL's[1], with a conditional expression, a then-clause and an else-clause. Set statements work by setting a value on the building simulator (in effect it acts as an actuator).

During the initial research for scripting languages we looked at several established embeddable scripting languages (see Figure 4.1).

Name	**Description**
Python	Python scripting
JRuby	Ruby scripting
Tcl/Java	Tcl scripting
Rhino	Javascript scripting
BSF	Bean Scripting Framework*

Table 4.1: Established scripting languages found during research

* The BSF supports several languages.

In the team we had some initial discussions if we should use tools that could assist us in designing our own DSL[2]. Examples of these kind of tools could be technologies like Xtext [4], Xbase [3] and EMF [1]. Xtext is a framework for developing programming languages and domain specific languages. Xbase is a partial programming language built in Xtext and used for integrating into other programming languages and domain specific languages. Xbase would be able to

[1] General Programming Language (like C++, Java etc.)
[2] Domain Specific Language

give us an working expression language. EMF is the Eclipse Modeling Framework, a modeling framework and code generation tool. Some of the team members in the danish group had used those technologies during the ITU course Model Driven Development, but no kenyan team members had any experience with these technologies. This would in effect mean that they would need to acquire the skills learned during Model Driven Development, while simultaneous working and contributing to our joint policy engine project. We deemed this unrealistic. Another argument for not using this set of technologies is that Xbase seemed too daunting a project to undertake due to it's current state. At the time of writing, only 25 questions have been tagged with Xbase on [2] - a website used frequently by the danish team members when in need of coding assistance. The lack of Xbase examples and documentation also reveals that it is a very new, and unfinished, product. Since Xbase would be the caretaker of the expression language, we could have opted to forego just that, and still use Xtext and optionally EMF. We considered this but arrived at the conclusion that using Xtext alone would not give us that much, since our language is small. Also the amount of overhead introduced by using Xtext, both in complexity but also in the generated code, would further add to the confusion of our team members with no experience with this technology.

Based on the these arguments we chose to develop the functionality ourselves. We also considered that task to be more *fun*! We believe that developing our own scripting language was best given the resources we had available and the timespan of the project. Considering the embeddable scripting languages (see Figure 4.1) we only needed a small subset of the functionality (mainly the if-then-else construct) and therefore deemed the complexity versus the timewise return of investment unfavorable.

4.1 Architecture

Merriam-Webster defines *policy* as "A definite course or method of action selected from among alternatives and in light of given conditions to guide and determine present and future decisions".

We have defined a policy as a collection of statements operating on sensors and actuators residing in the building simulator. The policy has a start time and a stop time, indicating when it is active. It is also possible to de-activate a policy completely — without deleting it entirely — using a boolean flag.

The system that we have designed consists of four overall parts;

- The management website (front end)

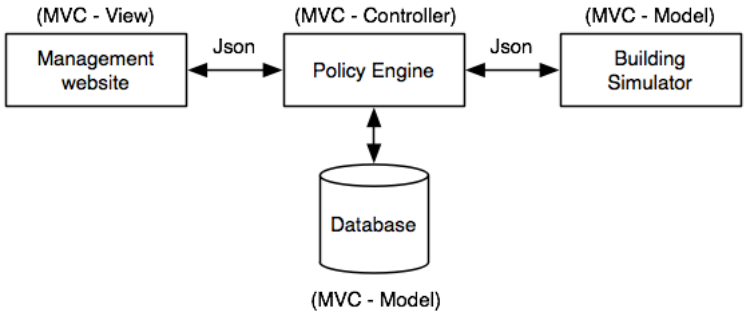

Figure 4.1: The PolicyEngine system architecture.

- The policy engine (back end)
- The database for storing policies (back end)
- The provided building simulator (back end)

Figure 4.1 illustrates the relationship of the system parts. It is evident that communication between both the front end and the simulator primarily is achieved through Json. For a more in-depth explanation, refer to Section 4.1.2.

In terms of MVC[3] the *model* and the *controller* is the servlet containing data and business logic. The servlet reacts to several urls (see Figure 4.2), that functions as an interface for an operation needed in the system. The *View* is the rendered html based on JQuery.

A statement can either be a SetStatement or an IfStatement. The SetStatement sets an value in the simulator. The backend implementation allows nested If-statements. If-then-else statements have been around possible before even the early versions of Basic, and is an established way of achieving conditional computer logic. Even though the intended audience of our system is not considered computer experts, the if-then-else concept is easy to understand and catches on fast when interviewing people. An If-statement can contain multiple expressions which are being anded when evaluated. If the user wants to make an If-statement with or'ed expressions, she will have to use a nested if. This raises the complexity and we do not consider this to be an optimal solution. The optimal solution in this case would have been to make a proper expression language that is recursion safe, for example based on [22]. This would also solve our lacking or's in the condition of the if-statements. Unfortunately we did not realize this until around two thirds into the development. Since our jQuery [29] parsing on the frontend naturally is strongly bound to domain classes on the backend (due to JSon serialization using Gson [10]), any big changes at that point would affect

[3]Model, View and Controller

both the backend and the front-end. We estimated then that it would be too time consuming to change the system. We therefore leave it up to future work (7).

4.1.1 The management website

The management website is a small collection of html pages that relies heavily on JQuery. The building policy administrator opens the main page in a browser, and gets an overview of all the policies and can perform CRUD[4] operations on them.

During the design phase for the GUI of the management website, we used Photoshop sketches that we mailed and transferred via Skype to each other. One from the kenyan team also assisted in this phase. After discussing how to best present policies to the building policy administrator in the time we had available, we choose to use both indentation and color coding page layout (see Section 5) when presenting the policies. By choosing this type of GUI, as opposed to a more tabular view on the if's, we elected a GUI that is flexible enough to extend to a recursive rendering which clearly matters a greatly when then-clauses can contain other if-statements that again contain then-clauses and so forth. For implementation see Section 5.

4.1.2 The policy engine

The policy engine consists of a servlet and domain classes for the expression language needed to build policies.

A statement can either be a SetStatement or an IfStatement. The SetStatement sets an value in the simulator. The backend implementation allows nested If-statements, making the policies both flexible and simple.

An If-statement can contain multiple expressions that all are being anded when evaluated. If the user wants to make an If-statement with or'ed expressions, she will have to use a nested if. The optimal solution to this would have been to make an expression recursion safe model. We did not have enough time for this, but we will elaborate further on this subject in the discussion (see **??**).

The statements are organized as a list. Each statement will be executed (set) or evaluated (if) in the order that they are persisted. Each if-statement has two lists that contain the statements belonging to the then-clause and the else-clause.

[4]Create, Retrieve, Update and Delete

The servlet is the core of our policy engine. It has three main objectives;

- The timed, repetitive execution of policies which currently are active
- Serving web service requests to the JQuery front end
- Serving the management website html files

An overview model of the PolicyEngineServlet can be seen in Figure 4.2.

The servlet container used is Tomcat 7.0.37[5].

This design was based on several iterations of group discussions. Everyone wanted a solution that worked efficient and fast, within the loosely defined scope that we initially had set for the system development. And clearly, our project should fit in the timeframe allowed for the project.

The first couple of design proposals from some team members were based on SQL integration — by the means of creating a java sql connection — directly into the database that the building simulator uses. This direct integration could be argued as being a good, although strongly coupled integration. Also, it did not feel like a real world scenario (which we are sure that ITU intends as being one of the reasons for this course) if we could access the database directly in this way. Traditionally building systems tend to be closed source proprietary systems with few, if any, integration point. Today there exists different kind of protocols (KNX, LON and BACnet to name a few) that allows some integration to existing building automation systems (for example CTS [6] systems that are widely used in Denmark). Since our project deals with a building simulator, we do not have to worry about these kind of implementation details — but we are aware that they exist.

[5]Other versions might work as well
[6]Central Tilstandskontrol- og Styring

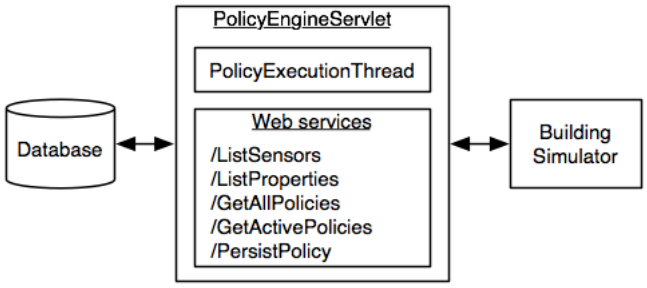

Figure 4.2: A overview model of the PolicyEngineServlet.

Later design proposals revolved around the idea of having more or less mirrored sensor data in a separate database. Data should be transferred at regular intervals, and the policies should be executed against the copy. This turned out to be a too unsophisticated solution, for several reasons. It would involve a lot of overhead copying data from one database to the other. There is also a problem in knowing exactly what data to copy, since that depends on the statements that is currently active. Based on the above mentioned discoveries we decided on the following design, that takes all team discussions into account and solves all earlier stated challenges.

On servlet container startup the servlet will read various configuration values, and then set up a thread that manages the timed, repetitive execution of active policies. While testing the system policies were executed every 5 seconds, but this is configurable. The servlet will start by querying the database for active policies, which are defined as policies that have a boolean flag (active) set to true, while the current time is within the policie's *from* and *to* time. This will result in a list of policies that should be executed. Then all sensor id values of all if-statements (including nested statements) are fetched from the building simulator, and cached in an in-memory datastructure. Then the policies are executed, and values set accordingly using the rest API.

4.1.3 The database

The database used is mySQL 5.5.29[7] and used for storing the policies. At the moment we only use one table containing all the policies.

4.1.4 The building simulator

The building simulator [13] provided in the course has not been changed or modified in any way.

[7] Other versions might work as well

Chapter 5

Implementation

This chapter is dedicated to the implementation of the presented system. In this chapter we will explain in detail what has been implemented and how.

5.1 System

The system has been implemented using Java and Servlets for the back-end, and JSP and JavaScript for displaying the webpages in the front-end. We have chosen to use MySQL as a database since it is known to all the members. The database is used to store the policy entities (the policy entity is explained in subsection 5.1.2). There is also a logging subsystem that is used to create logs about the system and its execution. This has been implemented using the log4j library and it creates a log file which contains all the logged data.
The back-end deals with requests from the front-end and executes at a specific time interval, the active policies at that time. The front-end displays the policies and enables the creation, modification or deletion of policies. However, because we have chosen to serialize the policies to JSON, these policies have to be deserialized and parsed to be displayed properly in the front-end.

A serialized policy to JSON has the following structure (see Figure 5.1)

It contains a statement object which is composed of the objects conditionalExpressions, thenStatements and elseStatements. The conditionalExpressions object describes the IfStatements and has information about the sensor id, the prefix operator, and the operator. The thenStatements and elseStatements have information about the desired value of the sensor and the id of it. For this example, the gain value of the light sensors on the floor 0 are compared to the desired value, 1 with the use of the "EQUALS" operator. If the gain value of a sensor is

```
{
  "statements": [
    {
      "type": "dk.itu.policyengine.domain.IfStatement",
      "data": {
        "conditionalExpressions": [
          {
            "prefixOperator": "AND",
            "aValue": {
              "type": "dk.itu.policyengine.domain.FloatValue",
              "data": {
                "floatValue": 1.0
              }
            },
            "operator": "EQUALS",
            "sensorId": "wildcard-floor-0-light-gain"
          }
        ],
        "thenStatements": [
          {
            "type": "dk.itu.policyengine.domain.SetStatement",
            "data": {
              "aValue": {
                "floatValue": 0.0
              },
              "sensorID": "wildcard-floor-0-light-gain"
            }
          }
        ],
        "elseStatements": [

        ]
      }
    }
  ]
}
```


Figure 5.1: A serialized policy

equal to 1, then the light is turned of by setting its gain to 0.

5.1.1 Interfaces

There are two interfaces that describe the methods of Statements and Value classes. Each statement should implement an execute method, in which specific execution is handled.

5.1.2 Domain

A statement can either be a SetStatement or an IfStatement. The SetStatement sets an value in the building server (in effect it is an actuator). It is possible to have nested IfStatements, making the policies both flexible and simple. An IfStatement can contain multiple expressions that all are being "*and*ed" when

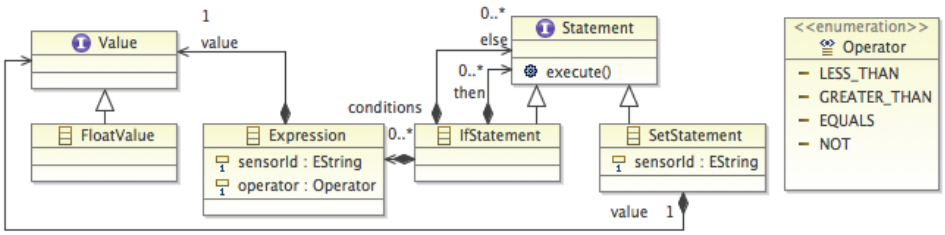

Figure 5.2: The core classes in the *expression language*.

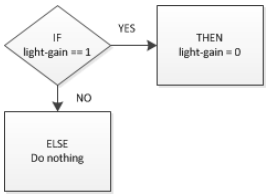

Figure 5.3: A concrete example of IF-THEN-ELSE based on the policy explained in Figure 5.1.

evaluated. If the user wants to make an IfStatement with or between the expressions, one will have to use a nested IF.

Using the example from 5.1, the IF-THEN-ELSE concept functions as shown in Figure 5.3.

An Expression has three variables, the sensorId, the operator and a value of type Value. When the policy engine is running, each expression is being evaluated. The current value of the sensor is fetched and compared to the desired value using the operator. An expression can contain the following operators; $<$, $>$, $<=$, $>=$, !=,=. If the evaluation of the expression is true, then the statement is executed.

The FloatValue class implements the Value interface and it represents a value of type float. This is used to set the values of the sensors, as those can be of float type. However, at this moment we only support to set integer values, 1 for On and 0 for Off.

A Policy object consists of statements and the policy can be run by executing those statements. A PolicyEntity object holds a policy and different properties of that policy such as name, description, interval, id and active. The interval property defines *from* which time *to* which time the policy should be enforced to run. This is implemented as a separate class, Interval, because of the serialization logics. The active property represents the status of the policy as in if the user wishes for this policy to be executed (active) or not (not active).

We have implemented wildcards to be able to handle many sensors and actuators. A wildcard defines the sensors and actuators handled, and their type property (gain, setpoint). An example of a wildcard is "wildcard-floor-0-light-gain". This expresses that we want to retrieve information about the gain of the light sensors from the floor 0. The ThenStatements and ElseStatements can be executed using the same wildcard, a different one or just some different sensors.

5.2 Persistence

We have decided to store our policies in a database as we think it is better to have them on a different server than the one running our policy engine. This may be a drawback if the internet connection fails to either servers. However, volatility is always a problem with distributed systems and this issue could be handled by having a backup database that resides on the policy engine's server and is synchronized with the main database server.
The database chosen was MySQL as it is well known and used by the group. Because we do not have a complex persistence system, we decided to use the simple JDBC for connecting and querying the database, and not use other specialized frameworks such as Hibernate[1].
Methods for communicating with the database are defined in the DataAccessLayer class. These are the methods for creating, reading, updating and deleting policies. In addition, it is of great interest to have a method that returns only the policies that are running at the current time and are active.
Different policies may operate on the same sensors, so we decided to use a cache to store each sensor's value. This reduces the number of queries to the building server so it does not overload it. This cache is implemented using a hashtable and stores the value of the sensor. After executing all the active policies at that moment, this cache is cleared.

5.3 Building querying

To communicate with the building's server, we created the Connection class which provides methods for connecting to it and parses the JSON object received. There are several methods implemented that parse the response of the server and offer information such as: a list with the sensors in a room by providing the room id, a list with all the sensors of a certain type (e.g. AC, light, heater etc.) The two most important methods are for retrieving the sensor's value and setting it to a specific one.

[1] `http://hibernate.org/`

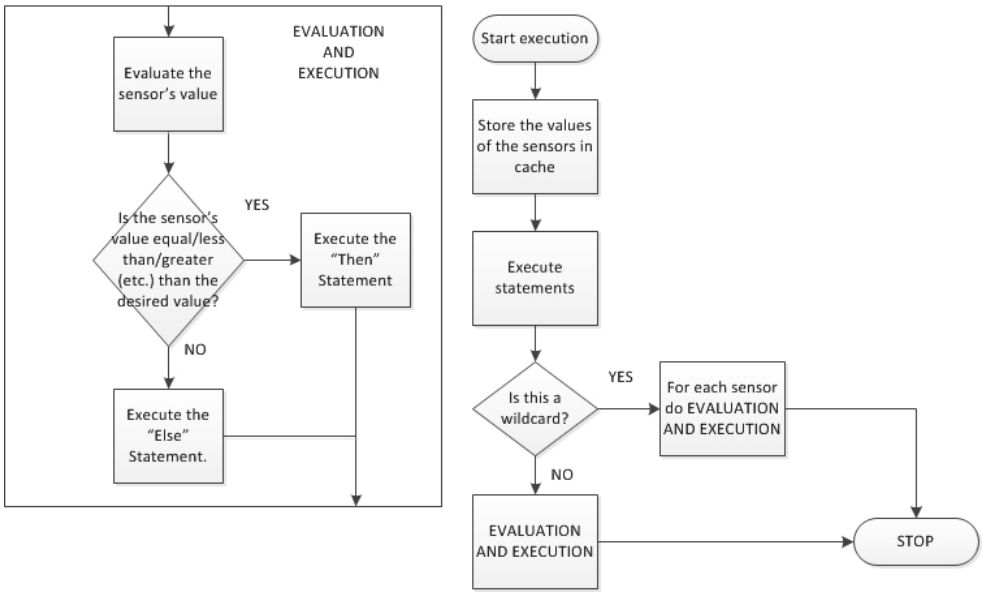

Figure 5.4: The execution flow of a policy.

To retrieve a sensor's value we use the GetSensorValue method which takes the sensor id as an argument and returns a string with the value. By default the values are sorted by timestamp, and the most recent value is returned.
We use the method SetSensorValue to set a value, which takes the sensor's id as parameter and the value we want to set.

5.4 Executing policies

The system executes the active policies at a specific time interval. This execution has the following flow (see fig. 5.4): First, a list with the active policies at that time is retrieved. Then the statements that compose the policy are retrieved along with their expressions. In these expressions resides the sensor id on which operations will occur next. For each sensor it is retrieved its last value from the building server and stored in the cache. After storing all the values of the sensors, the statements are executed.
The first statements that are executed are the IfStatements. Here a check is done to see if the policy is a wildcard. If it is a wildcard, then the statements are executed for each sensor by evaluating its value against the desired value. If the evaluation returns false, then the else statements are executed. If the policy is not a wildcard, an evaluation of the sensor's value is done and depending on its result, the thenStatementes or elseStatements are executed. In the evaluation part the actual value of the sensor is compared to the desired value using the

operator defined in the expression. This evaluation returns true or false and so execution of the thenStatements is done if the evaluation is true otherwise the elseStatements are executed. In the execution of the then- and else-statements, the new sensor value is set in the building's server.

5.5 Front-end

5.5.1 Initial Ideas

We quickly realised that our ideas could easily expand the project, with numerous functionality and features, that would bring the project to a far too complex level. We mapped all our ideas and prioritized them, and made a selection of ideas, which we planned on implementing.

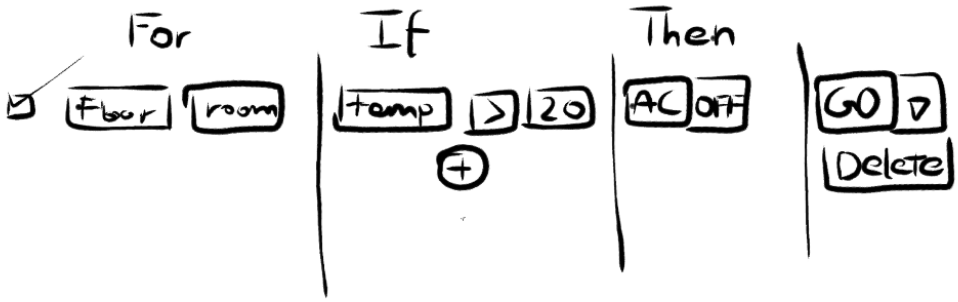

Figure 5.5: Initial drawing.

The selected ideas that we wanted to realise (sorted by importance) included:

- Users should be able to add, delete and modify policies.
- Users should be able to use complex operators in policies, e.g. higher than, lower than, equal to etc.
- Users should be able to save policies and easily toggle an ON or OFF state.
- Users should be able to combine multiple sensors in a single policy.
- Users should be able to use wildcards in a policy, e.g. effecting for instance an entire floor without the need to specify the rooms that belong to the chosen floor.
- Users should be able to create nested rules in policies.

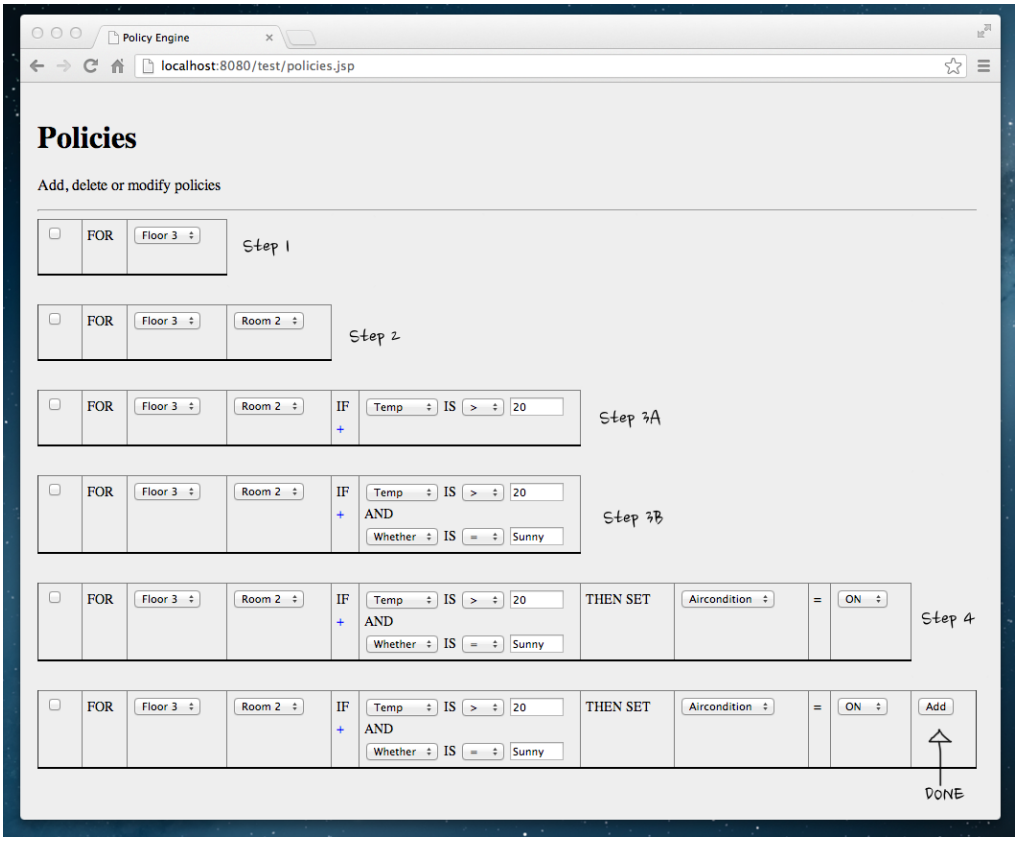

Figure 5.6: Initial drawing - step wise build up.

All these points where realised in the backend of our implementation, however because of time limits we regret that the possibility to create nested statements did not make it to the user interface. Still we have an interface fully ready to support nested statements, but enabling this feature is something we have added to future work instead. We have started to work on a recursive function to be able to cleverly support endlessly (in theory, computing resources are not endless) deep nested statements. These nested statements would be listed under THEN and would contain a new set of IF, THEN, ELSE subsections. These sections would be easy readable because of indentation and encapsulation.

5.5.2 User Interface

The main goal for the user interface has been to keep it simple and user-friendly without loosing advanced functionality. Initially we made rough drawings to better understand what we were dealing with and how to best possibly present it. Doing so we tried to sketch the process of handling a policy from the creation

to the enabling of it. We came up with various approaches, one of them being the step wise (see figures 5.5, 5.6). However this approach proved to be not as flexible as we desired and hard to handle very detailed and complex polices. Our aim has been to make a great overview for the user. Therefore we chose another concept that in which we split each statement up into an open / closeable area to better encapsulate the logic. We believe that it is easier to read one statement at a time, therefore you can keep open the important statements and closed the other ones. Within each statement we also decided to split up IF, THEN and ELSE into subsections because you in theory can add as many entries to each subsection as needed. Nonetheless the introduction of Wildcards allows for manipulation of entire floors, taking up much less space and work for the user to define than having to set up entries for each sensor or actuator on a floor. See an example from the final view of a policy build at figure 5.9.

5.5.2.1 Intuitive Design

What you want to achieve when designing an interface is to make it as intuitive as possible. This can be archieved by looking at the user's "knowledge space" [5].

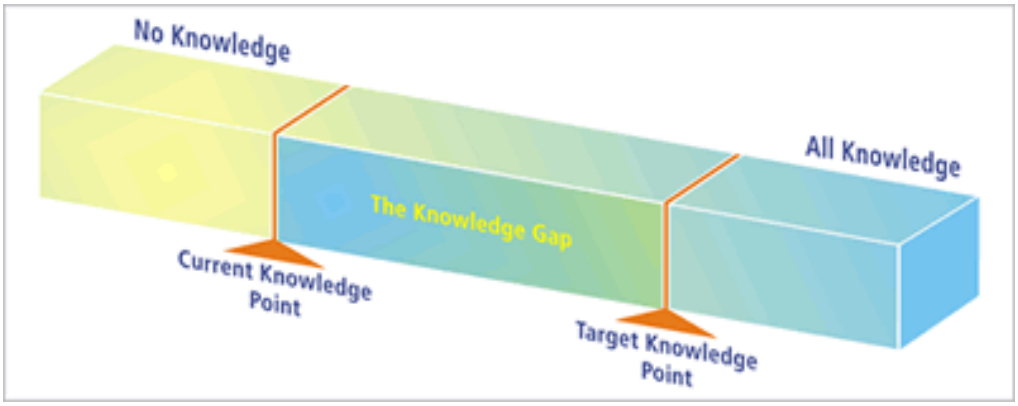

Figure 5.7: The space between the Current Knowledge and Target Knowledge points is called The Knowledge Gap.

The distance from the left (As seen on figure 5.7) indicates how much a given user is familiar with an interface, this is called the "current knowledge point". The next point of interest is called "target knowledge point". This section details how much knowledge the user needs to have to achieve their goals. Each time a user attempts to perform a specific task, their "current knowledge point" and "target knowledge point" are very important for us to identify. Each user's "current knowledge point" and "target knowledge point" will be different when they gain more experience. However, it turns out that, by making usability tests we can

often identify groups of users who have almost the same "current knowledge point". This is also why we have performed think-aloud tests presented in 6.2.

"The Knowledge Gap", which is the distance between the "current knowledge point" and the "target knowledge point", is also called "The Gap". "The Gap" is what you have full focus on designing for. There is no need to design to the left of the "current knowledge point" and you do not need to design to the right of the "target knowledge point" since the user does not need the knowledge to perform the task. Therefore the focus should be only on designing the interface for the space in between the two points.

Users can accomplish their goals when the "current knowledge point" is equal to the "target knowledge point". There are two ways to achieve this. One way is to train the users, thereby increasing their current knowledge, until they know everything they need to do the job. The other way is to design an interface that can reduce the "current knowledge point" by making the interface more accessible until the "target knowledge point" requires only the information the user already has.

5.5.2.2 Flow

We have strived to create an interface that provides a good flow to the user. The user should not notice the interface but have full focus on the task to be solved - the interface should be transparent. Any distraction breaks the flow. [6]

> "No matter how beautiful, no matter how cool your interface, it would be better if there were less of it." - Alan Cooper

There are three prerequisites for achieving flow:

- The user must have a clearly defined goal.
- There must be a good balance between the user's skills and challenges the user must deal with.
- There must be clear and immediate feedback.

Especially the last point of the 3 above can be difficult to achieve, when you are in an online environment with response times. However, we believe that we have solved this by allowing the user to model their policy without any server interaction until they want to save.

5.5.2.3 Response Times

In order for the user to get the best possible experience on a website, it is important to optimize response times. Below is the overview of user's reactions at different response times from Alan Cooper - About Face 3:

Up to 0.1 seconds
Users perceive the systems response to be instantaneous
0.1 – 1.0 seconds
Users notice a delay, but their thought processes stay uninterrupted
1.0 – 10 seconds
Users regard the system as slow, and their mind is likely to wander
After 10 seconds
The system will loose the users' attention. They start doing other things or switch to another application

Based on this it is important to optimize the web site, both for the code generated on the server, but also to optimize the front-end since around 80% of the load time [27] of a website is spent on parsing the generated HTML document, execute javascript and download any external resources. There are a number of things you can do, which we will not go in detail with but just touch on. A few points we are aware of is where and in what order javascript and css is located and loaded, javascript can be loaded last as javascript usually is loaded after a page is fully parsed. Furthermore, you can set preferences for cache and reduce the number of external files to load in an application by combining various javascript files into one and also combine CSS files.

5.5.2.4 Color Coding

Color can be a powerful tool to improve the usefulness of an information display in a wide variety of areas if color is used properly. Conversely, the inappropriate use of color can seriously reduce the functionality of a display system [31].

We are using color contrasts to express information. We have used this concept to color the different boxes under a statement to improve the readability.

We have taken into account that red-blue and yellow-blue color combinations are generally safe for color blind people. Having this in mind can lead to a much higher ability to use color coding effectively. This will still cause problems for those with monochromatic color blindness [16], but it is still something worth considering.

5.5.2.5 Responsive Design

Responsive web design is a web design approach aimed at crafting sites to provide an optimal viewing experience, easy reading and navigation with a minimum of resizing, panning, and scrolling across a wide range of devices (from desktop computer monitors to mobile phones). While we haven't directly optimized for mobile we have had the responsive design idea in mind when setting up the design. What we have assured is that the policy editor is adjusted to the screen size of the user. Meaning if you for instance use a full HD resolution (1920x1080) the interface will utilize this space and boxes with expressions will position themselves accordingly (see example on figure 5.9).

5.5.3 Policy Editor

5.5.3.1 Managing Policies

In order to make it simple and fast to manage a policy we have split up the interface into sections and sub sections. There are two sections: Policy Information and Policy Statements. In Policy Information, all the content: name, description and time interval are defined. Also the policy can be set to be active or not in this section. The Policy Statements section is where all the expressions are constructed. We have added a sub section for each statement to improve the readability and overview of what is executed and what is paired together (see figure 5.8).

Within a statement there are three sub sections: IF, THEN, ELSE (see figure 5.9). For each headline in this setup there is a "+ New" button that can be clicked. This will add a new entry to this particular area. While this is done we alter the policy Javascript object in the background so once the user is satisfied with the setup of statements, it can be saved with a click on the update / save button.

To help the user find the correct sensor / wildcard option, we have implemented an auto-complete functionality. This lets the user start typing a value and then the auto-complete will come up with all the matching elements available.

To make it take less space and work to define a statement that should apply to an entire floor or room, we have introduced wildcard options. All wildcard options are manipulating more then one element all start with "wildcard". A use case for a wildcard can for instance be: A user wants to work with the blinds on the entire first floor: wildcard-floor-1-blind-setpoint is then the option to pick.

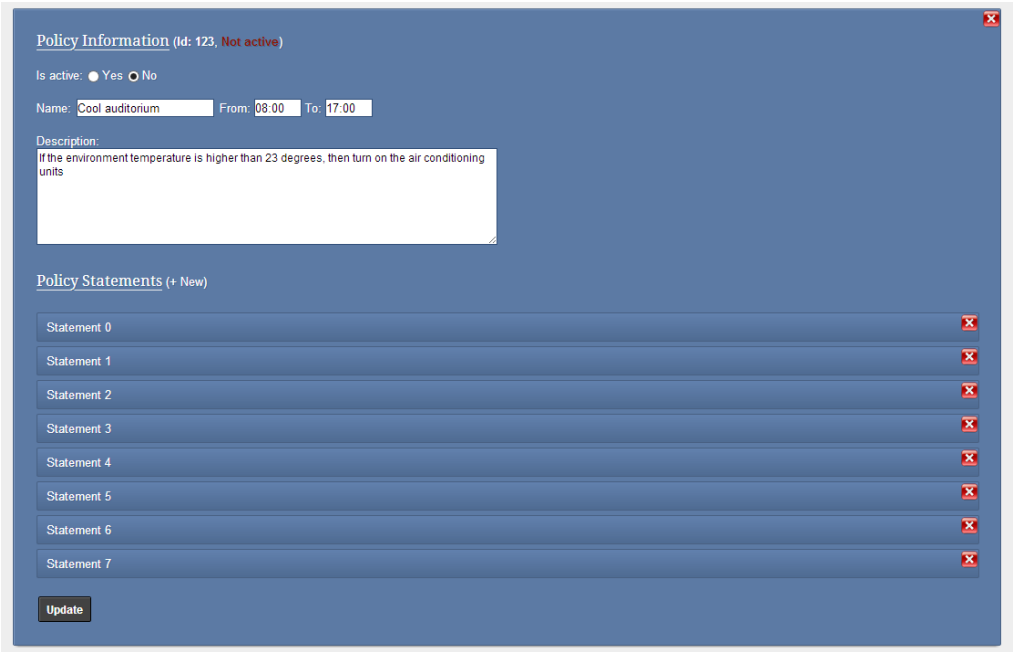

Figure 5.8: Editing a policy.

5.5.4 Use Case View

The solution's final use case diagram is shown in Figure 5.10.

The user starts on the policy list view. Here the user have two possible paths to take: Modify a policy or create a new. Within a policy you can modify details, enable / disable it, setup expressions or delete it.

5.5.5 Usability

Seeing policies as rules for rooms in a building, with multiple sensors in each and every room, that can affect multiple actuators - not only in one room, but all of the rooms in a building: It leaves the user with a lot of selections. This will, if not represented properly, complicate the process of adding new policies.

Following the principles of Steve Krug's "Don't Make Me Think: Common Sense Approach to the Web"[19], the functionality of a website should always let users accomplish their intended tasks as easy and directly as possible. Throughout the development of the front end site we have strived to do so. And we have had two usability tests (more on this in the Evaluation part) to uncover potential problems.

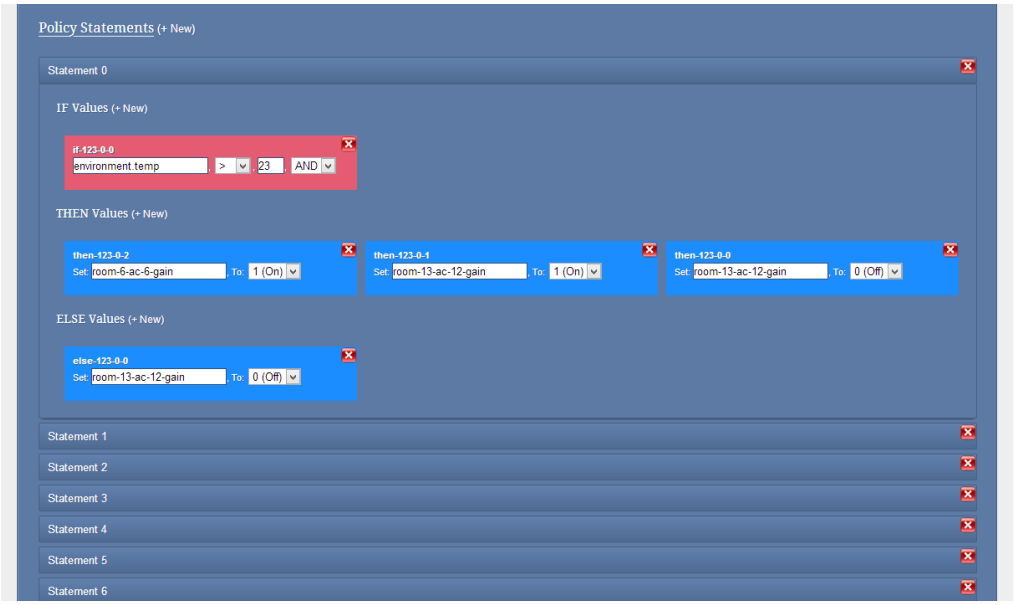

Figure 5.9: Configuring statements.

5.5.5.1 Browser Compatibility

We have optimized for modern web browsers. The specific browsers we have tested our solution in are: Internet Explorer 9 and 10, Firefox, Chrome, Safari.

5.5.5.2 Layers of Abstraction

We have focused on the Visual UI layer enabling a graphical representation of the CRUD (Create, Read, Update Delete) process. Another approach is having policies handled in a textual manner, and thereby forcing users to a write the statements in a policy, using a domain specific language. For a developer textual editing and console commands might be preferred, and may somewhat be quicker, while they know the inputs by heart. But the intended users of our policy engine, are most likely not developers or fans of typing in commands.

In a future version of the policy engine, both methods could be implemented to function in parallel, giving a user both choices, and also the capability of copy and paste policies quickly.

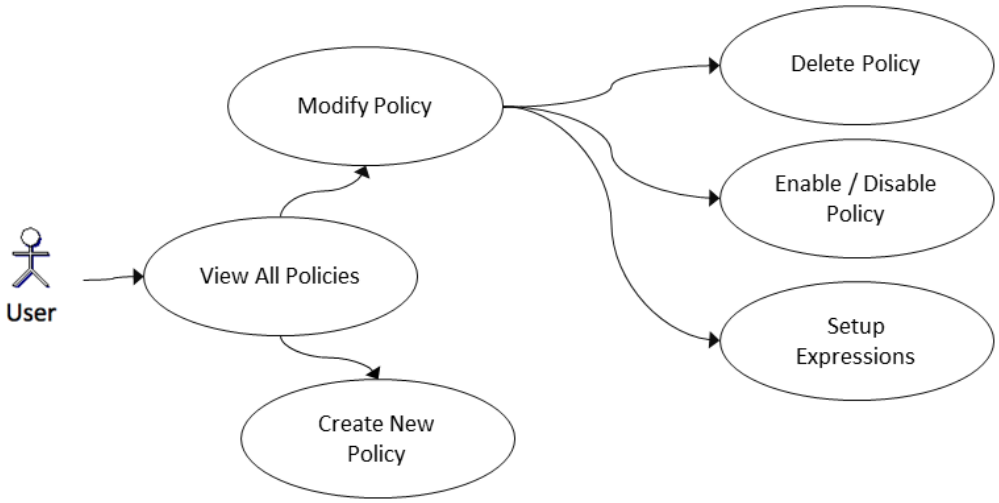

Figure 5.10: Use case diagram.

Chapter 6

Evaluation

We have evaluated two main areas;

- The functional solidity — using log files and JUnit [17] tests
- The front-end usability — using the Think Aloud Protocol

Clearly, both areas are important for a successful software system.

6.1 Functional Solidity

JUnit tests are structured, assertable software tests that ensures that the software behaves as expected. Of course TDD[1] — which we employed using the development of the core functionality — has it's limits, mainly with regards to the quality of the written tests. The JUnit test accounts for the low level implementation tests.

On a higher abstraction level, we employ log files for testing. A log file is being generated at run-time, and has been integrated into (amongst other areas) the expression language. An example; we already know a certain policy's expected behavior (because we defined it ourselves) — then we make sure that the policy is executed by the policy engine, and after the successful execution we carefully study the log file and compare it with the expected behavior. We find the log tests a natural selection on top of the more low level automated JUnit tests. It gave us a sense of extra security in regards to the behavior of the policy engine.

[1] Test Driven Development

6.1.1 Log Testing

We have logged system data which was important as to determine if the system was working correctly. Doing so we have been able to verify the behavioral results and to discover bugs in our implementation — which we then iteratively corrected.

We have also been using the log files for source code quality control. We verified our solution by cross-referencing every action during tests, and matched it with the actual data stored in the database.

One excerpt from the log file can be seen in figure 6.1

```
2013-05-11 23:56:28,122 [Thread-1] INFO  dk.itu.policyengine.domain.Expression - Value is 0.0
2013-05-11 23:56:28,122 [Thread-1] DEBUG dk.itu.policyengine.domain.Expression - Is 0.0 equal to 0.0 ?
2013-05-11 23:56:28,122 [Thread-1] DEBUG dk.itu.policyengine.domain.Expression - Yes.
2013-05-11 23:56:28,122 [Thread-1] DEBUG dk.itu.policyengine.integration.Connection - Trying to connect to
http://gsd.itu.dk/api/user/building/entry/set/1/room-14-heater-14-gain/1/?format=json
2013-05-11 23:56:28,242 [Thread-1] INFO  dk.itu.policyengine.integration.Connection - {"returnvalue": true}
```

Figure 6.1: An excerpt from the log file.

6.1.2 JUnit Testing

To strengthen the solidity of the software, and minimize time used on debugging, we implemented a suite of JUnit Tests. An snippet of JUnit code can be seen in 6.2. Entering this test, the *ROOM1_HEATER* has been set to 1 and *ROOM1_TEMPERATURE* has been set to 26. The purpose is to see if the underlying expression language will set the *ROOM1_HEATER* to 0, which it should because the Expression's condition is using the *Operator.GREATER_THAN* along with a value of 25.

We have employed a sufficient amount of JUnit tests, to test all operators and both IfStatements (including nested capabilities) and SetStatements.

The result was that our IF-THEN-ELSE concept implementation performed as expected.

6.2 Usability Test

By using the Think Aloud Protocol we tested to uncover potential usability issues that needed solving.

To evaluate on the usability of our policy engine we decided to make a test with participants outside of our development group. In general, when it comes to

```
@Test
public void simpleGT_Execute() {
    Expression expression = new Expression(ROOM1_TEMPERATURE, Operator.GREATER_THAN, new FloatValue(25));
    Statement thenStatement = new SetStatement(ROOM1_HEATER, new FloatValue(0));

    IfStatement ifStatement = new IfStatement();
    ifStatement.addExpression(expression);
    ifStatement.addThenStatement(thenStatement);

    Assert.assertEquals(1f, SensorValueCache.getValue(ROOM1_HEATER).getValue());
    ifStatement.execute();
    Assert.assertEquals(0f, SensorValueCache.getValue(ROOM1_HEATER).getValue());
}
```

Figure 6.2: A snippet of a JUnit test.

"best practices of usability tests", the Think Aloud Protocol is considered one of the most valuable [23].

The Think Aloud Protocol is time wise inexpensive and easy to set up and it also gives very valuable result from real-life scenarios.

6.2.1 Think Aloud Protocol

In a Think Aloud Test, you ask test participants (one at a time) to use the system while continuously thinking out loud — that is simply verbalizing their thoughts as they move through the user interface, and take actions.

To run a basic Think Aloud usability study, three things are required:

- Recruit representative participants.
- Inform them of representative tasks to perform.
- Avoid interference and let the participants speak their actions.

We invited five people to each Think Aloud Test. Research shows that having just five people will potentially uncover 80% of all usability problems [24] as seen on 6.3.

We conducted two Think Aloud Tests, these tests were aimed at fixing any usability issues found during the third iteration.

We have not directly targeted Facility Managers in our usability tests, as we simply did not have the time to gather enough test persons with a relevant job function. However instead we gathered a broad range of random test persons with different backgrounds and age.

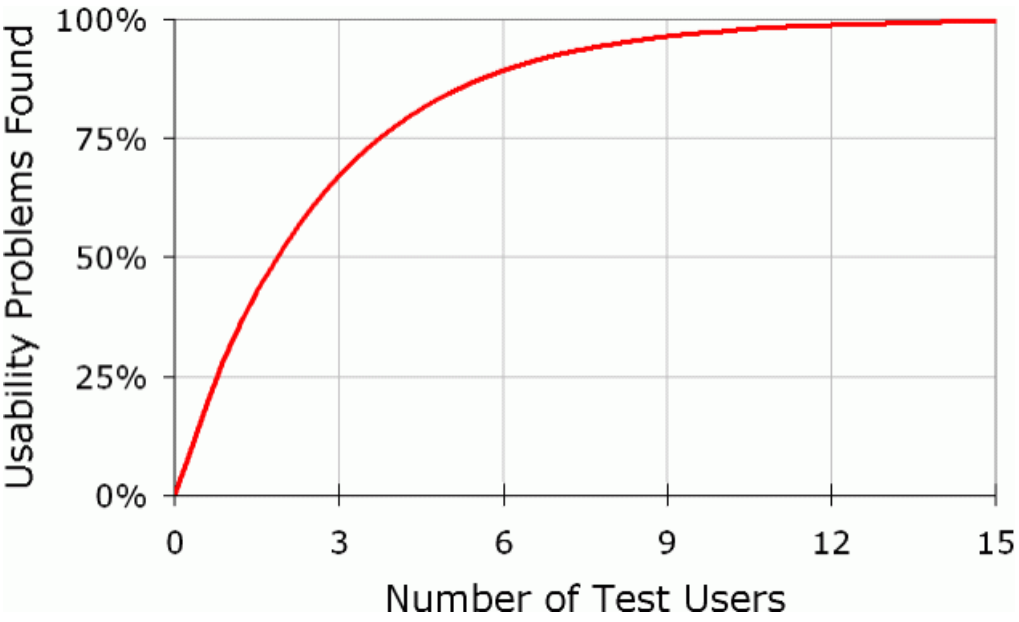

Figure 6.3: Usability test graph.

6.2.1.1 Tasks

We arranged 10 tasks for the participants to perform. Note however that the tests where held individually, but they were all given the same set of tasks. During the tests, participants where only given one task at a time to focus on - using small task cards.

Before the test started we explained the purpose of the policy engine, and verbally gave them an example of a user scenario.

We briefly introduced the participant to a map of the building and the rooms involved. We also gave an example list of sensor and actuator names to make them familiar with the elements involved, including basic knowledge of IF, AND, THEN statements and wildcard operators. The questions were designed so that the test persons used all of the features available in the policy engine. The user interface contain an *live auto-complete function* so that the participant can easily set a desired sensor/actuator without knowing the exact name. See section 5.5.3.1 for more details.

The participants where assigned the following tasks:

Create a policy that turns the AC on in Room 1, 1. floor and name it "Cooling".

The 1st task was designed to see how the participant would try to create new policies - and how they would set the name.

> Modify the "Cooling" policy just created to affect both Room 1, 1. floor and Room 2, 2. floor.

The 2nd task was designed to see how the participant handled modifying an existing policy.

> Create a policy that turns on blinds in all rooms for every floor in the building and name the policy "Sun".

The 3rd task was designed to see how the participant handled the "wildcard" auto-complete feature.

> Find and show the active policies just created.

This 4th task showed how well the participant handled the listing of active policies — as this would likely be a typical reoccurring task for a building administrator.

> Disable the policy named "Cooling" so that it is no longer in function.

The 5th task showed how well the participant handled enabling/disabling of a policy.

> Delete the policy named "Sun".

The 6th task showed the removal of policies.

The tasks 7-10 were designed to be harder and the participant was forced to work with complex expressions.

> To save energy the university wants to have the heating turned OFF automatically at 17:00. However this Wednesday around 19:00 - 22:00 an exclusive presentation is held in room number 5 on 1. floor. You are asked to maintain a temperature at 21 degrees in that room throughout the presentation.

It is summertime and the overall temperature inside the building is rising. You are asked to keep the temperature at maximum 22 in all of the rooms in the building.

All afternoon between 12:00 and 16:00 the sun is at its peak. Therefore you are asked to set blinds ON in all the rooms on 1. floor and 2. floor - but only if the lights are ON.

The university wants to save energy. You are asked to make sure lights are automatic turned OFF at 17:00 in all the rooms, except from those on 0. Floor.

6.2.1.2 Results and Comments

After every participant had gone through the Think Aloud Test. We went through all their "thoughts" and issues. Some were the same, these we combined into one. For all the remarks, we made some comments as listed below — including the actions we took to further improve the software.

> If there is a lot of statements - which I would expect there will be? Then I think the statement list would be too long. I think it would make it difficult to keep the overview, if say you have 100 statements.

The participant were referring to the “detail view” of a policy, from which every statement were listed underneath each other.

Due to this remark we have implemented an expand function, which instead of listing all the content at once, it now only shows each statement headline as an open / closable area.

> I think it is difficult to see which one (rule) belong to what (expression).

The participant were referring to the setup within a given policy. When a policy contains an IfStatement, the list of conditional expressions, and the then-clause and else-clause created some confusion. The participant found that it was difficult to separate the different parts.

Due to this we implemented the aforementioned color coding 5.5.2.4.

In the second Think Aloud Test with five new persons, the result showed that they found the overview better and easier to understand after implementing this encapsulation and color coding.

After the second round of testing we only had some minor changes to do. But overall the participant where comfortable with the usability and had no further remarks.

We believe that there is always room for improvement, and we elaborate on this in the section 7.2.

Chapter 7

Discussion

7.1 Collaboration Evaluation

In this section we discuss the overall project and the collaboration details. For a technical discussion see 4 and 5.

The collaboration proved to be a rather challenging task. Only one person from Strathmore showed any interest in the project, even though our initial expectation was a four man team. We believe that the lack of interest was not caused by our approach towards the students from Kenya but either; 1) a lack of interest and willingness to work on the project or 2) bad luck with the student selection - they might be swamped in work or having too many demanding courses.

We believe we did all we could in regards to motivating our foreign colleagues. In the beginning we sent them an e-mail to introduce ourselves, based on the TA's experience regarding past collaborations with students from Kenya. We asked them in a kind way to update our "contact information" document, so we could reach each other on Skype/chat/email etc. The last member from the Kenyan group updated the contact information after 16 days - and thereafter was not heard from again. We created a mailing list (gsdall@netstarter.dk) with all our email addresses to ease the communication between all the members. All the emails were written in English, even when communicating between Danish team members. We wrote out status messages, at least once every week. During the length of the project, we only heard from two of the Kenyans. One of them counted for almost all communication, and he willingly became the *team leader* for the Kenyans — though that did not help on their work efforts. We also tried to schedule Skype meetings, on our weekly full working day (Tuesdays). However, only one Kenyan was there, and he was only online around half the Tuesdays. And often it was very late (16-20:00) until he came online, while the

Danish team members had been at it since 10:00 AM (12:00 PM in Kenya). Perhaps this would have worked better if we would have switched time for the meetings, but unfortunately this was not possible due to overlapping times with other courses.

In order to improve our collaboration, we would have liked to create sub-groups of participants mixed from Kenya and Denmark, and assigning those groups a task. However, we never managed to set it up, due to the lack of commitment from the Kenyans. These subgroups would have been useful for the implementation part, where different tasks would have been assigned to subgroups, making it easier to communicate and deliver. Instead we decided to make two subgroups in the Danish team, one working on the front-end and one on the back-end, while keeping the Kenyan group informed about our progress, and motivating them to write about their efforts. Approximately one month before deadline, we only focused on involving the one active Kenyan in some GUI developments. The others would have been invited to participate also, clearly, but they never came online or answered any of our 30+ status emails. Through the elected Kenyan team leader we were told that either they did not have the time to participate or that it was too hard to get online. One of the members also quitted the course, due to personal reasons.

It has become clear to us that we could not have done much differently in regards to the collaboration. Two members have directly responded that they do not wish to participate in the project, one is not actively participating and the last group member is highly unreliable. These motivational challenges are core values that one needs to have to participate in a global software development project. No strategies or collaborative approaches could have transformed the situation into something positive. These challenges are not a part of a normal work setting, thus one probably would be reprimanded by not attending meetings, for not replying to emails and for not showing any motivation towards the work.

If the initial motivation was there, a set of different methods could have been used to kick off a great collaborative team. Some of them are discussed in 3. One approach that could have been useful was to introduce each others countries and talk about topics not directly related to the core project. We tried opening up for this kind of communication early in the project, by asking the one active kenyan guy to help us make a dictionary between our two languages. He explained there are many languages, but Swahili is the official language. As a result we made a "Swahili - Danish" dictionary document. The active kenyan team member helped filling out different words, and the idea was then that we should try to speak a little in each others language when Skyping. Unfortunately, due to the general lack online kenyan team members, we never really used it. The few times we actually communicated with them via voice, it would have been too “forced” and strained to use that approach. We would have liked to communicate more and

more often with our kenyan team members. This way one would get to know the different team members and know more about them. This could have been done by just simply introducing one self, family life, interests, hobbies and the likes.

We suffered a lot from the lack of contact with the Kenyans, especially during the first iteration. Our hopes and expectations for the collaboration was high, and thus we wanted our Kenyan team members to give us feedback and inputs on our early choices of programming platform, setup and the likes. We probably waited too long time for their opinions, which caused a delay in our project schedule. We could have solved this by creating a group contract that one had to obey, and if not, one was not part of the group any more. This was discussed during a meeting by the Danish team members, but it would have been a ultimatum - put forth mainly due to frustration and not to improve the collaboration. We could have used more strict deadlines and structured the work even more. This way we would have an early indication of their motivation, or lack hereof, and would be able to push the project forward without their involvement.

It should have been possible for us to directly contact the assigned Kenyan teacher, and asked him to contact the students and have them explain to him what hindered their participation. We have been informed that the intermediaries between ITU and Strathmore University was the TA's[1] and the course manager. We tried several times to address this issue, but either the intermediaries did nothing to elevate the issue at Strathmore University or, more likely, it did not have any effect. If ITU is going to do a collaborative project with Strathmore University again, we would advice that ITU implements much tighter teacher to teacher communication - resembling employer to employer conversations held in private companies when venturing into a partnership. We believe it should have had a consequence that a 5 man team effectively only is an unreliable 1 man team. We expect that the Global Software Development should somewhat resemble real world setting. We find it unrealistic implementing such an idea without any leverage mechanism.

We also believe that it could have made a change if the Universities coordinators agreed upon a specific kick off date, where the team members would meet online and coordinate the work from here. This would ensure that we at least got to meet all of our team members, and had some basic knowledge about each other.

We decided to assign different switchable roles to each person in the Danish group. The Kenyan team leader was at one point also assigned a role because he one day participated almost a whole Tuesday. The roles switched during the project, but our focus was to achieve the highest performance throughout the process. The roles were assigned through an analysis of each person's set of skills, motivation and interest. Some persons focused more on the backend, while

[1] Teaching Assistants

others worked on the frontend, and others on the report, as their primary tasks. This has showed to be a useful tactic because of two different reasons; a) first of all we know that the assigned person is motivated to work with the assigned area and secondly b) he has the skills, or wants to learn the skills, to solve the problem at hand. This should not be interpreted as a person was "stuck" doing only one thing. Team members could easily shift to other areas, and that did also happen.

Another useful tactic was our weekly meetings, with a simplistic agenda: What is the current status and how can we push the project forward? All of our communication during these meetings were very specific and all related to these challenges and how to solve them. The benefit was that everybody was up to date with the current progress, knew the challenges and had the possibility to participate in finding a solution. These statuses were also emailed through our group email, so people not attending had the possibility to catch up.

7.2 Future Work & Improvements

Though we have tried to make the policy engine as flexible as possible, some issues remain. For example, if we are having a policy that rolls down the blinds during the night, and we want them to roll up when the policy is no longer active (decided by the toTime property), it is not possible at the moment — when just using a single policy. We can of course create a new policy that rolls up the blinds at a certain time in the morning.

As explained earlier, our expression language should be improved to allow multiple expressions with the choice of OR as opposed to the current AND-only functionality. With the GPL's[2] on the market today, people expect to be able to make complex conditions — even though the users of the system are not IT experts. This could be accomplished by extending our expression language so it is recursion safe, or embedding an expression language inside our policy engine that is already recursion safe. The aforementioned nested statements should also be a possibility to create within the GUI. While we have taken the preliminary steps for this we did not have time to fully implement it.

We would also like to have made some enhancements to the SetStatement functionality, in particular *temporal constraints — Time* and *State*. Explained shortly, *Time* should be used in two ways. It should be able to conditionally query if the current time is within some specified interval. This could be used for making policies behave differently on certain times (for example, from 14:00 to 19:00 on Wednesdays). Time can be added relatively simple, as a basic-edition wrapper

[2]General Purpose Language

of the GregorianCalendar class. *States* are thought to be boolean variables accessible globally and locally - dependent on which scope it was declared. This way, for example, a fire detection policy could set the *buildingIsOnFire* state, and each other policy would be able to change it's behavior if needed based on access to that state. States can also be programmed easily, by having a map or a hashtable with key and boolean value pairs. The local states can be implemented by using the same structure for the global states, but with the policy id as a prefix in the name. Due to the project's time-constraints — and the four missing resources that should have been available from Kenya — we decided to forego of this nice-to-have functionality and focus our efforts elsewhere.

Another improvement could be to have a policy analysis that validates the expressions. For instance in our implementation there can be two policies that manipulates the same elements, but one is doing one thing while another is doing another thing. Currently in the implementation what ever expression is executed last wins.

7.3 Threats to Validity

7.3.1 Internal Threats

Selection: One of the internal threats are the group composition. We did not have any influence in who we were paired with from Kenya. The motivation, abilities and experience did prove to be very different within the two subgroups.

Experimental Mortality: Differential loss of participants across our group proved to affect the end result. We had three out of four Kenyans drop out. This put a lot more work on our shoulders and forced us to cut down on some features for instance the nested statements in the GUI.

7.3.2 External Threats

Chapter 8

Conclusion

This project focused on developing an online platform for managing sensors and actuators in a building, with students from Denmark and Kenya. The difficulty of this project consisted of working in a global distributed team and providing a policy engine prototype. We presented our functional prototype which is able to create, modify, and delete policies. The prototype executes the policies against the provided building simulator. We have designed the prototype with usability and flexibility in mind, and it aims to be used by people that are not IT experts. This resulted in having a simple and friendly interface for the users.
Another goal of this project was the collaboration with Kenya but this collaboration has not been a succes for several reasons. We could have done things different but in the end the people from that group did not commit at all for this project. Though our prototype works as expected, there can be several improvements. One feature that we would like to have is temporal constraints in the Statement mechanism. This would provide even more flexibility in defining policies. Another improvement would be to employ an analysis mechanism of running policies, which would detect possible conflicts (setting different values on the same actuators) between running policies. We thank Matias Bjørling, Javier González and Aslak Johansen for their feedback and offered support on the building simulator.

Bibliography

[1] Eclipse modeling framework. http://www.eclipse.org/modeling/emf/. Online; accessed 20-April-2013.

[2] Stack overflow. http://stackoverflow.com/. Online; accessed 20-April-2013.

[3] Xbase wiki entry at eclipse.org. http://wiki.eclipse.org/Xbase. Online; accessed 20-April-2013.

[4] Xtext home. http://www.eclipse.org/Xtext/. Online; accessed 20-April-2013.

[5] What Makes a Design Seem 'Intuitive'? Internet. http://www.uie.com/articles/design_intuitive/, January 2005. Online; accessed 10-May-2013.

[6] Alan Cooper, Robert Reimann, and David Cronin. About face 3: the essentials of interaction design. Wiley, 2007.

[7] D. Dietrich, D. Bruckner, G. Zucker, and P. Palensky. Communication and computation in buildings: A short introduction and overview. Industrial Electronics, IEEE Transactions on, 57(11):3577–3584, 2010.

[8] Folketingets EU-Oplysning. Internet. http://www.euo.dk/fakta/tal/internet/, August 2011. Online; accessed 7-May-2013.

[9] Facebook. Green of facebook. https://www.facebook.com/green, April 2013. Online; accessed 2-May-2013.

[10] Google. Gson. https://code.google.com/p/google-gson/, April 2008. Online; accessed 2-May-2013.

[11] Google. Google green. http://www.google.com/green/, April 2013. Online; accessed 2-May-2013.

[12] Dae-Man Han and Jae-Hyun Lim. Smart home energy management system using ieee 802.15.4 and zigbee. Consumer Electronics, IEEE Transactions on, 56(3):1403–1410, 2010.

[13] ITU. Gsd2013 building simulator. `https://gsd.wikit.itu.dk/GSD2013+Building+Simulator`, April 2013. Online; accessed 11-May-2013.

[14] Rod Janssen. Towards energy efficient buildings in europe. n/a, 2004.

[15] Sirkka L Jarvenpaa and Dorothy E Leidner. Communication and trust in global virtual teams. Journal of Computer-Mediated Communication, 3(4):0–0, 1998.

[16] Joseph Carroll Jay Neitz and Maureen Neitz. Color vision: Almost reason for having eyes. Optics & Photonics News, 1047-6938, 2001.

[17] Junit. Junit. `http://junit.org/`, April 2013. Online; accessed 14-May-2013.

[18] Andrew Krioukov, Gabe Fierro, Nikita Kitaev, and David Culler. Building application stack (bas). In Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings, BuildSys '12, pages 72–79, 2012.

[19] Steve Krug. Don't Make Me Think: A Common Sense Approach to the Web (2nd Edition). New Riders Publishing, Thousand Oaks, CA, USA, 2005.

[20] Piritta Leinonen, Sanna Järvelä, and Päivi Häkkinen. Conceptualizing the awareness of collaboration: A qualitative study of a global virtual team. Computer Supported Cooperative Work (CSCW), 14(4):301–322, 2005.

[21] Jian Li, Jae Yoon Chung, Jin Xiao, J.W. Hong, and R. Boutaba. On the design and implementation of a home energy management system. In Wireless and Pervasive Computing (ISWPC), 2011 6th International Symposium on, pages 1–6, 2011.

[22] Robert C. Moore. Removing left recursion from context-free grammars. In Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference, NAACL 2000, pages 249–255, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[23] J Nielsen. Iterative user interface design. IEEE Computer, 26(11):32–41, 1993.

[24] Jakob Nielsen. Why you only need to test with 5 user. `http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/`, March 2000. Online; accessed 7-May-2013.

[25] Evelyn Njoroge. 14m kenyans have access to the internet. http://www.capitalfm.co.ke/business/2012/01/14m-kenyans-have-access-to-the-internet/, January 2012. Online; accessed 7-May-2013.

[26] Gary M Olson and Judith S Olson. Distance matters. Human-Computer Interaction, 15(2):139–178, 2000.

[27] Part 1: What the 80/20 Rule Tells Us about Reducing HTTP Requests Performance Research. Internet. http://www.yuiblog.com/blog/2006/11/28/performance-research-part-1/, November 2006. Online; accessed 10-May-2013.

[28] Microsoft Publishing. Microsoft environment. http://www.microsoft.com/environment/, April 2013. Online; accessed 2-May-2013.

[29] John Resig. jquery. http://jquery.com/, January 2006. Online; accessed 2-May-2013.

[30] Philip Haves & Thierry Stephane Nouidui Simulation Research Group: Michael Wetter. Building controls virtual test bed. http://simulationresearch.lbl.gov/projects/bcvtb/, April 2013. Online; accessed 2-May-2013.

[31] J.M Taylor and G.M. Much. The effective use of color in visual displays: Text and graphics applications. Color Research and Applications, 11(3):3–10, 1986.

[32] A. Tiberkak and A. Belkhir. An architecture for policy-based home automation system (pbhas). In Green Technologies Conference, 2010 IEEE, pages 1–5, 2010.

[33] Mary Beth Watson-Manheim, Katherine M Chudoba, and Kevin Crowston. Distance matters, except when it doesn't: Discontinuities in virtual work. n/a, 2007.

[34] Yao-Jung Wen, Dennis DiBartolomeo, and Francis Rubinstein. Co-simulation based building controls implementation with networked sensors and actuators. In Proceedings of the Third ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings, BuildSys '11, pages 55–60, New York, NY, USA, 2011. ACM.

Appendix A

Appendix

A.1 How to Access the Policy Engine

During this project the individual team members have developed and tested the webapplication locally on their own computer using Tomcat Apache 7.0. However a public accessible solution have also been deployed on an Ubuntu server instance in the cloud[1].

To access the solution please go to `http://46.137.101.196/gsd-pe` in a modern web browser.

A.2 How to Access the Source Code

For the purpose of easy access and collaboration, the team have created a Github account to hold the source code.

To access the source code please go to `https://github.com/tkok/GSD-code` or see the files located on the CD attached to this report.

A.3 Project Plan

The project plan below is a Gantt Chart. The chart shows our milestones and deadlines.

[1] We are using Amazon EC2 for this purpose

Figure A.1: Gantt Chart of the Project Plan

A.4 Doodle Schedule

The Doodle schedule shows at which time each participant was able to be online in a collaborative manner. The schedule was only used to create an initial idea of which days each member were available to work on the project. The green boxes shows availability and red boxes unavailability. Each member selected between three time slots for each day in a week. 8AM to 12PM, 12PM to 4PM and 4PM to 8PM. See figure A.2.

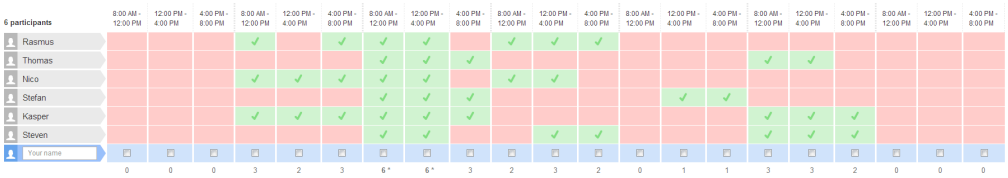

Figure A.2: Our Doodle plan showing a normal week from Sunday to Monday. Everybody is available to work each Tuesday.