A Distributed, Collaborative Development Project of a Policy Engine for Use in Buildings

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 3, 2013

Abstract

This paper presents the process of the development of a policy engine platform. The team behind the platform consists of members from respectively Strathmore University, Kenya and IT University, Denmark.

The motivation for doing this project is to dive into the world of distributed collaboration between to teams in different countries, while working on an real-life project. The field of building automation has seen a lot of growth in recent years. Building automation can save a lot of energy and resources by e.g. turning off light when nobody is in the room and the likes. This could possibly be a massive improvement in countries such as Kenya where there is a lack of resources. It would allow the resources to be used in the right conditions and not just wasted.

The general approach to the project is to solve the aforementioned policy project while collaborating with team members across continents. First steps is to create a fundamental team platform to communicate from. Afterwards we will focus on the technical project.

Our result is a fully working prototype of a policy engine platform, where one can create, edit and de-/activate policies. Our evaluation shows that... **SOMETHING FROM EVAL.**

Lastly we show that our approach towards building a policy engine is valid.

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 and the likes. An example of a policy could be to turn off heating at midnight. Manually controlling an energy-saving policy is a time-consuming and inefficient task and the user have to know a lot of different equipment as well as information about the building. It 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 buildings facility management[1]. The task of controlling the policies are 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 of the building benefits from the automation provided. If policies are correctly defined, building owners save energy and other natural resources while providing extra comfort to their tenants.

Building users can experience a building autonomously adjusting its internal environment to suit their comfort and needs, while FM can achieve fine-grained control of the building.

As stated in [5] *Worldwide, there is no doubt that efficient energy saving is only possible with modern building automation, known as BA, based on networking in all levels of abstraction.*

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

The development focus of this project is therefore geared towards this simulator. The end product is a web-based management console, that allow for centralized flow control between sensors and actuators in the simulated building. This is achieved by implementing the policy engine that allows for automated actuator responses based on sensor feedback. An example at this would be closing the blinds in excess sunlight or turning of the heater when the 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.
4. Document the collaborative project between the IT University of Copenhagen, Denmark and Strathmore University, Nairobi Kenya.

[1]from now on FM

1.1 Context

Modern buildings get ever 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 [10] [7] [22].

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[2] and AC's[3] 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 - which makes our task extremely 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 it's 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 (by issuing http post commands with sensor id and the desired value) continuously.

[2]heating, ventilation, and air conditioning
[3]Air Conditioning

1.2 Problem

The task of controlling the buildings gets likewise progressively more complex - especially if that control should be optimized in regards to both economic and human comfort. 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 building subsystems.*

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 to collaborate with team members distributed in different continents and countries, with many differences such as cultural and temporal.

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.

One could summarize the learning goals to be: 1. How to work together in a distributed team and reflect on this and 2. How to develop a policy engine that conforms to the stated requirements (see Requirements).

1.4 Requirements

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

START COPY/PASTE FROM https://gsd.wikit.itu.dk/Policy+Engine

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%).

END COPY/PASTE FROM https://gsd.wikit.itu.dk/Policy+Engine

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 [26], where a Policy Based Architecture for Home Automation System is developed. The system is composed of five 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.

Another approach is the one taken by Li et. al [18] 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 [13] 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 exists for the light service, heating and air conditioning. In [29] develop a prototyping platform based on Building Controls Virtual Test Bed framework [25] 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.

Chapter 3

Method

In this chapter, the workflow or working process used throughout the entire project is described. Furthermore we present how the collaboration was performed and which strategies we used.

3.1 Workflow

In our design of a policy engine, we decided to use an iterative design approach, both for the front-end and back-end. Using this approach we were able to continuously evaluate both the concept and our software. Throughout the development, we have done a lot of informal testing. The goal have been to continuously test and optimize the software and learn about the stability and performance and adjust the implementation accordingly.

For the front-end design an iterative design approach is commonly used. This allows designers to identify any usability issues that may arise in the user interface before it is put into wide use. Even the best usability experts cannot design perfect user interfaces in a single attempt, so a usability engineering life cycle should be built around the concept of iteration. [20]

We ended up doing three major iterations:

3.1.0.1 1. Iteration

3.1.0.2 2. Iteration

3.1.0.3 3. Iteration

3.2 Collaboration

Before any actual work could start, one preliminary goal was to figure out how we could make our group work together as one. Actually this challenge is even more so in this project than in a normal work situation: No organization is in order, no predefined roles, no actual project goals and the likes. This chapter will focus on these challenges and how we tried to handle them.

3.2.1 Social Context

When we discuss the *Social Context*, we discuss the direct milieu in which the person is and how different factors can influence this person. Communication is also a part of the social context, between one to many persons, in different time zones, different cultures etc.

3.2.1.0.1 Common ground Our first step to connect to our fellow group mates in Kenya was to introduce ourselves via an e-mail and just shortly highlight some common information about each person from ITU, like stating name, age etc. This method is known as creating "common ground", as introduced by Olson and Olson [21]. 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" [21]. Common ground is not only established through simple general knowledge about each participant. It is also created through a persons behaviour and appearance through meetings and conversations. This is often created between persons with the same temper, sense of humor and the likes. We tried to use this method as a way of getting to know our team members, to create a level of understanding and finally to create a stepping stone from which the project could evolve from. The method of creating common ground is an early way of creating a feeling of a unity, and getting to know each other.

3.2.1.0.2 Trust and First Impression This initial contact was already quite frustrating because of the fact that it was difficult to get a reply from some of the group members in Kenya, only two members was relatively easy to get in touch with. This leads directly to two different considerations in our group work: “Trust” and “First impressions matters” as introduced by Jarvenpaa et al. [15]. Trust in group work is, as the term might indicate, a value of how much the different team members trust in each other. How much does one believe that the other team members will deliver their part of the necessary work? How much does one believe that a mail will be answered? How well does the team work together? The trust between the two subgroups, Denmark and Kenya, relatively low because of the amount -and lack of- replies and general communication. At the time being we only expect one member from Kenya to be online during our team sessions but at the same time we expect everybody from the ITU-group to be online at every session. Trust is an important factor in a group setting; It’s a foundation and a crucial part to solve. The group work process is highly related to how well the group function. This is also directly coherent from the first impressions that we received from the group from Kenya, with the lack of communication and willingness to participate in the project.

3.2.1.0.3 Collaboration Readiness The literature for these challenges seems to agree that these sort of problems generally arise from two different topics. One being “Collaboration- and Technology Readiness” and the other “Continuities/Discontinuities”. The latter part will be discussed in the subsequent section, Groupware Technologies (3.2.3).

Collaboration readiness is a potential show stopper for the team work, if a given member is not ready to collaborate. This could be caused by having conflicts in interest, e.g. one is about to overtake another persons job or the likes. This could cause that the person, who is about to lose his job, not to be ready to collaborate in a productive manner. 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.

One thing that could cause their lack of interaction in our e-mail correspondences and Skype meetings are the technology readiness. We know that it is a challenge for some of the group members to get internet access. A recent study shows that 36,3 % has access to internet in Kenya [1], which compared to Denmarks 86 % in 2010 [2], indicates that it is not as easy to get access as we are used to in Denmark. And we know by fact that a few of them does not have a stable internet connection in their home. Our approach to solve this issue was to have a meeting each Tuesday at 10:00. This way we know that they should have access

[1]According to http://www.capitalfm.co.ke/business/2012/01/14m-kenyans-have-access-to-the-internet/

[2]http://www.euo.dk/fakta/tal/internet/

to their University, which most of the time has an internet access they could use. We know they should have time for this meeting, thus it is planned as a course on their schedule.

3.2.1.0.4 Ethnocentrism A potential threat to the well-being and harmony of the group is known as Ethnocentrism [6]. Ethnocentrism is a state of one subgroup, where the members sees that one group as the centre of everything, and every other group will be valued and ranked from this. A subgroup is a group inside a group, some of the group members are part of - but not the whole group is part of. This way it could create some sense of “Us versus Them”, which is something you definitely want to avoid. One way to avoid this is to create multiple subgroups inside the group. One subgroup could be those who prefers to work with Java and backend-programming, while others prefers jQuery and frontend-development. This could be subgroups across the different countries. If you are part of multiple subgroups the feeling of being part of just one group will dissolve, which should result in a more harmonious group. This way of creating subgroups would be something that you did early on during the initial communication, and something we tried to solve by writing small parts about each other member from Denmark. Unfortunately, we did not receive any feedback from Kenya. This has probably strengthened the feeling of us vs. them, because we do not know much about them. We definitely have a feeling that our group is the centre right now due to the fact that it is only the Danish group that develop and contribute to the project.

3.2.1.0.5 Coupling of work Coupling of work relates to the state of the current tasks and how loosely or tightly coupled they are. A completely loosely coupled work is one you can perform without the interaction and feedback from other persons. A tightly coupled work task is, on the other hand, one you only can perform with other members of the group participating. Our project has evolved from a very tightly coupled project to a more loosely coupled. This was done on purpose. In the beginning of the project everybody had to be at the meetings because of the fact that we had to define which way to move the project. The development life cycle was quite rigid and strict. After the initial phase, where we decided on the platform, chose an architecture etc., more and more tasks became slightly more loosely coupled one. This means that one could start to work on his part of the project without any direct interaction with other team members. This would also allow such a highly distributed team as ours to collaborate in an efficient way. It would just be too inefficient if everybody had to be together at the same time and place every time anything regarding the project should happen. As of right now we communicate through different Groupware Tools (see 3.2.3) and only meet through one weekly meeting.

3.2.2 Collaborative work

Collaborative work across cultures is a challenge. In our case we had to work together 9 people. 4 being from Kenya, a culture and country that we before entering this project, knew little about. As mentioned earlier the social context and the process of creating “common ground” with the collaborators is of high importance. The goal was to create a fundamental shared understanding of the task and build up the motivation and the much needed trust for the collaboration to succeed. Cooperative work is defined by Schmidt as “People engage in cooperative work when they are mutually dependent in their work and therefore are required to cooperate in order to get the work done” [24].

3.2.2.0.6 Articulation work Articulation work is the extra activities required for collaboration [24]. The extra work is the essence of everything that is needed in order to fulfil the task, e.g. coordination of tasks. Articulation work is about who does what, when and where. There are mechanisms of interaction that supports the process -the life cycle of the project- when articulation work cannot be handled through every day social interaction. These mechanisms are for instance: Organizational structures (formalinformal), plans, schedules and conceptual schemes. Our strategy for the articulation work was to define processes and choose the groupware technologies that supported our cause best possible.

3.2.2.0.7 Coordination The more spread out between people and artefacts a task is the more the reach is increased. Increased “reach” of a task changes the coordination. Segregation, which is a method to separate a complex tasks to smaller subtasks, is a suitable strategy when a complex task is at hand. By dividing the complex task into smaller tasks you can often solve them concurrently and thereby complete the larger complex task faster. It also allows for more specialised teams to investigate a task at a more detailed level. Therefore we identified a given number of subtasks, by segregating the larger task. This could be to identify tasks such as “Create DataAccessLayer”, instead of an overall task like “Create Backend”. This is one of the ways of how we segregated the tasks within our project. Afterwards each group member were assigned to these tasks, who would work closely together, until the given subtask was completed.

To handle these tasks we created a project plan with deadlines and milestones. Moreover we agreed on having a status meeting every week, where we would discuss progress, issues, ideas etc. Arranging a meeting where all are able to attend is not always easy. We did manage to meet on Skype at times which took into consideration both the differences in time zones (temporal discontinuities, see 3.2.4.0.11) and the fact that people had entirely different classes and work schedules.

With computer supported cooperative work (CSCW), it's impossible to anticipate every contingency. There will always be exception handling. The core challenges and dimensions of cooperative work includes articulation work, adaptation of technologies and awareness. The lack of trust and awareness when you never meet face to face with your collaborators is problematic and requires methods and training when using communication tools. We primarily used Skype to communicate with and made sure to document changes and commits to the code base very detailed. Individuals working together need to be able to gain some level of shared knowledge about each other's activities.

3.2.3 Groupware Technologies

One way to describe collaborative software, is that it is a software designed to help people achieve a common work task within a group. One of the earliest definitions of collaborative software is found in the research paper "Rhytms, Boundaries, and Containers" by Peter and Trudy Johnson-Lenz. They describe the software as: "intentional group processes plus software to support them." [16]. The definition was first stated in 1974. Many things have changed, especially in the IT business, but the way a group collaborates and the group dynamics has not. Some general terms are still vital to discuss. We find the most importantly used terms in our project to be critical mass, adaptation and adoption process. These three terms and a general overview of the used tools will be explained in the sections below.

3.2.3.0.8 Tools Generally we used four different tools to communicate our teamwork and progress; one synchronous and three asynchronous. We used Skype for every team meeting and for general communication between the members of the group. Skype is synchronous communication as you will normally get an instant reply while using voice chat. We furthermore used three asynchronous communication methods, that is GitHub, Email and Google Drive. GitHub was used to share the actual projects current status by communicating via tickets what needs to be done. Email was used for communicating general group announcements, that we want to make sure that all of the members from the group receive. Lastly we used Google Drive to share important documents.

3.2.3.0.9 Critical mass Critical mass [11] is a sociodynamic that describes the necessary amount of people needed in order for a product to be a success. We have used this quite directly in our usage of groupware technologies, such as Skype and Google Drive. After selecting which tools to use to communicate between the different members, we have made a great deal out of actually using the tools. This has created the critical mass, for each tool, and every member knows that they can reach each other through these, and thus made it necessary for one to use.

3.2.3.0.10 Adaptation & Adaption Process Adaption and the adaption process, by Tyre and Orlikowski, [27] are two closely related topics that we briefly touched through our project. These terms relates to how well an organization adapts a new technology and how they do it. Some of the problems with new technologies are that human beings tend to have routines that are not that easy to steer away from. This results in a thought of "Why should I use this new technology, when it works perfect the way I used to do it". One of the ways we tried to solve these problems was that we chose our tools as a group, and not by enforcing it. This way the majority chose their favourite tool. The adaption process describes how an organization adapts a new tool: The first couple of weeks are extremely important for a successful adaption. If the adaption does not succeed through these weeks it is very likely that the tool will fade out and never be used again. Our approach to this challenge has been that we pushed it to the people who did not know it, and tried to help them with the setup etc. An example of this could be the installation of Eclipse. One of the members from Kenya had some problems, and we all stayed online throughout the process and helped him install it. This way he could push it to his fellow team members in Kenya.

3.2.4 Virtual Project Management

Virtual Project management is about handling the entire project online, in a virtual and decentralized manner, where the users can access the project from any place and contribute seamlessly in developing the solution. We coordinated activities through GitHub: handling the code base, reporting and updating issues as we moved forward. The mode of interaction has mostly been through chat and voice via Skype, and not through face to face meetings. This presented various discontinuities.

3.2.4.0.11 Discontinuities Watson-Manheim et al. [28] examine virtuality in terms of boundaries and discontinuities. They define discontinuities as "a break or gap in the work context", or a "lack of continuity". They proposed the concept of discontinuities as a general notion to permit a more comprehensive understanding of the many ways in which virtuality can be perceived. Distance is the most obvious boundary that is encountered in virtual work but it is clear that there are more boundaries such as time, organization, and nationality, which are not usually present in more conventional work settings to the same extent.

It is only when those working in virtual settings perceive a boundary to be a discontinuity that it hinders work processes.

General properties of discontinuities are that they can emerge and change over time as people adapt in the teams. Discontinuities may only affect parts of the work. The typical discontinuities are temporal (working across time zones), geographic work location, work group membership (e.g. who you work with), organizational affiliation and cultural backgrounds. However discontinuities can also be expertise related (novice versus experts), historical (different version of a product), different professions (e.g. developers and researchers) or different technologies.

In our group we experienced how the different cultural backgrounds became a discontinuity. After a couple of scheduled meetings it was clear to us that the respect towards group agreements were not as important in Kenya as in Denmark. This became especially clear in regards to scheduled meetings, where the Danes often would only be a couple of minutes late, if any, which was not the case with the Kenyan members. This could be seen as a general cultural difference, or just the few members different respect of these meetings.

Another cultural difference we encountered was in regards to their view on us as Europeans. When asking Kenyans for feedback or criticism they are often fine with how it is. This might be because they do not want to cause problems and do not want to appear impolite. This could also be a part of their culture as a way to show respect.

The concept of teams varies across cultures and organizations, and how teams are perceived will differ based on the organizational and national cultural attributes of its members, according to Gibson et al [8]. It is also critical to note that individuals from different national cultures vary in terms of their group behaviours and communication styles, as Gudykunst specifies it [12].

3.2.4.0.12 Continuities Continuities are the opposite of discontinuities. Continuities are the stable factors in the collaboration that the participants are aware of and consciously act upon, or they may be implicit and unrecognised [28]. Often continuities can be described as strategies or factors to overcome discontinuities.

Chapter 4

Design

During our brainstorm sessions it quickly became clear that we needed a policy engine that was flexible. This we deducted from the cases discussed amongst team members. With inspiration from other building management systems (see Related Work) and looking at the web service we had to communicate with, we came up with the concept of policies that consist of to types of overall statements; *if-statements* and *set statements*. If-statements works as if-statements in many popular GPL's, with a condition consisting of expressions, 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).

4.1 The policy expression language

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 also has a start time and a stop time. It is also possible to de-activate a policy completely - without deleting it entirely.

A statement can either be a SetStatement or an IfStatement. The SetStatement sets an value in the simulator (in effect it is an actuator). The backend implementation allows nested If-statements, making the policies both flexible and simple (XXX how can we claim that?). 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

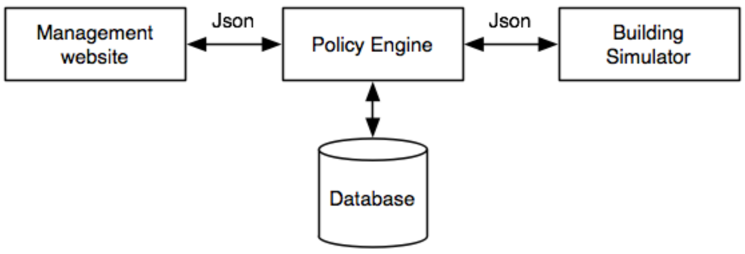


Figure 4.1: The PolicyEngine system architecture.

optimal solution to this would have been to make a safe left-recursive model. We did not have enough time for this, but we will elaborate further on this subject in the discussion (XXX remember to elaborate).

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.

4.2 Architecture

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

- The management website (front end)
- 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. In terms of MVC the *model* and the *controller* is the servlet containing data and business logic. The servlet reacts to several urls (see XXX), 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 (in effect it is an actuator). The backend implementation allows nested If-statements. If-then-else statements have been around since 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 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. 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 left recursion safe, for example based on [19]. 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 [23] parsing on the frontend naturally is strongly bound to domain classes on the backend (due to JSon serialization - using Gson [9]), any big changes at that point would affect both the backend and the frontend. We realized then that it would be too time consuming to change the system. We therefore leave it up to future work. We will elaborate further on this subject in the discussion (XXX remember to elaborate).

4.2.1 The management website

The management website resides is based in 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 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 an indented page layout 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 XXX.

4.2.2 The policy engine

The policy engine consists of a servlet and domain classes for the expression language needed to build policies. The servlet is the core of our policy engine. It has three main objectives;

- The timed, repetitive execution of policies that 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.

As servlet container we use Tomcat 7.0.37. 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 profiles, 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 are fetched from the building simulator, and cached in an in-memory datastructure. Thereafter the policies are executed.

This design was based on several iterations of group discussions. Everyone wanted a solution that worked efficient and fast, within the scope and timeframe that we initially had set for the system development. (XXX reference to project setup, project plan, scope, limitations?).

The first couple of design proposals from some team members were based on SQL integration 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 assume is one of the points of 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 (XXX reference to for example CTS systems).

Later design proposal 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 - and the problem to decide which sensor data to mirror remained. Finally we arrived at the rest api integration for fetching

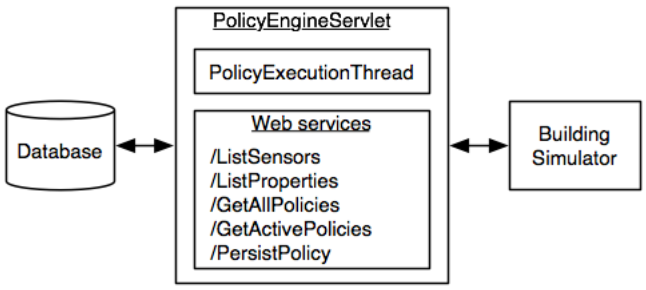


Figure 4.2: A overview model of the PolicyEngineServlet.

and setting values, and our own database to hold the policies and their time information.

4.2.3 The database

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

4.2.4 The building simulator

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

4.2.5 Limitations & alternatives

The core functionality of our policy engine is the expression language. During the initial research for scripting languages we looked at several established embeddable scripting languages.

Name	**Description**
Jython	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.

As stated earlier, we believed that developing our own scripting language was best given the resources we had available and the timespan of the project. It would propose a great number of challenges to integrate one of the scripting languages into our code. We only needed a small subset of the functionality (mainly the if-then-else construct) and we based our decision on developing our own language on the timewise return of investment.

In the team we had some initial discussions if we should use 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 give us an working expression language. EMF is the Eclipse Modeling Framework, a modeling framework and code generation tool. Even thought some of the team members in the danish group had used those technologies during the ITU course Model Driven Development, the current state of Xbase seemed too daunting a project to undertake. 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. However, we chose to develop the functionality ourselves. We also considered that task to be more *fun*! Developing our own scripting language made us think about how languages are designed, and the role of parsers, compilers and interpreters.

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.

5.0.6 System

5.0.7 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 done.

5.0.8 Domain

A statement can either be a SetStatement or an IfStatement. The SetStatement sets an value in the simulator (in effect it is an actuator). It is possible to have nested IfStatements, making the policies both flexible and simple. An IfState-

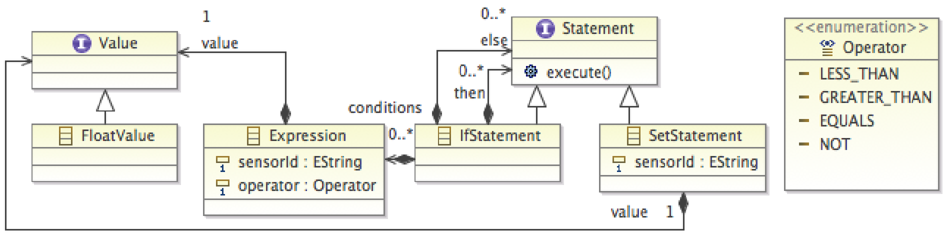


Figure 5.1: The core classes in the *expression language*.

ment can contain multiple expressions that all are being anded when evaluated. If the user wants to make an IfStatement with or between the expressions, will have to use a nested IF. The optimal solution to this would be to make a safe left-recursive model. We did not have enough time for this, but we will elaborate further on this subject in the **??** section. 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.

5.0.9 Persistence

We have decided to have our policies stored in a database. The database chosen was MySQL as it is well known and used by all the members. 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 any frameworks. Methods for communicating with the database are defined in the DataAccessLayer. There are the methods for creating, reading, updating and deleting policies. In adition, it is of great interest to have a method that returns only the policies that are running at the current time. Because the fact that different policies may operate on the same sensors, we decided to use a cache to store each sensor's value. This reduces the number of queries to the simulator server so it does not crash. This cache is implemented using a hashtable. After each iteration of the running policies, this cache is deleted.

5.0.10 Request Flow

5.1 Front-end

5.1.1 Initial Ideas

The main idea for the user interface is to keep it simple and user-friendly without loosing advanced functionality. Initially we started making a rough drawing - so to better understand what we were dealing with (See figure 5.2). Doing so we tried to plan ahead on how the visuals could look like, the functionality, and how users would be able to control the policy engine.

We quickly realized that our ideas could easily expand the project, with numerous functionality and features, that would bring the project to a far more complex

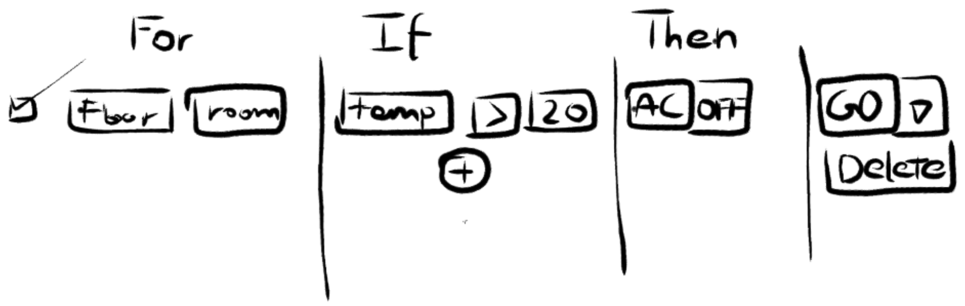


Figure 5.2: Drawing the initial idea of the element needed on the website, so a user can create, delete or modify a policy.

level. We mapped all our ideas and prioritised them, and made a selection of ideas, which we planned on implementing.

The selected ideas included that:

- 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 create nested rules in policies.
- 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 save policies and easily toggle an ON or OFF state.
- Users should be able to combine multiple sensors in a single policy.

The ideas were broken into three main front-end pages[1], from where the user can control the policy engine and overlook its operation.

The three mains are:

1. A site where a user can create, delete or modify -policies.
2. A site that visually represent the building digitally with floors and rooms (like a map used to navigate through rooms and floors).
3. A site that displays sensor values and group them in regards to the individual room they represent.

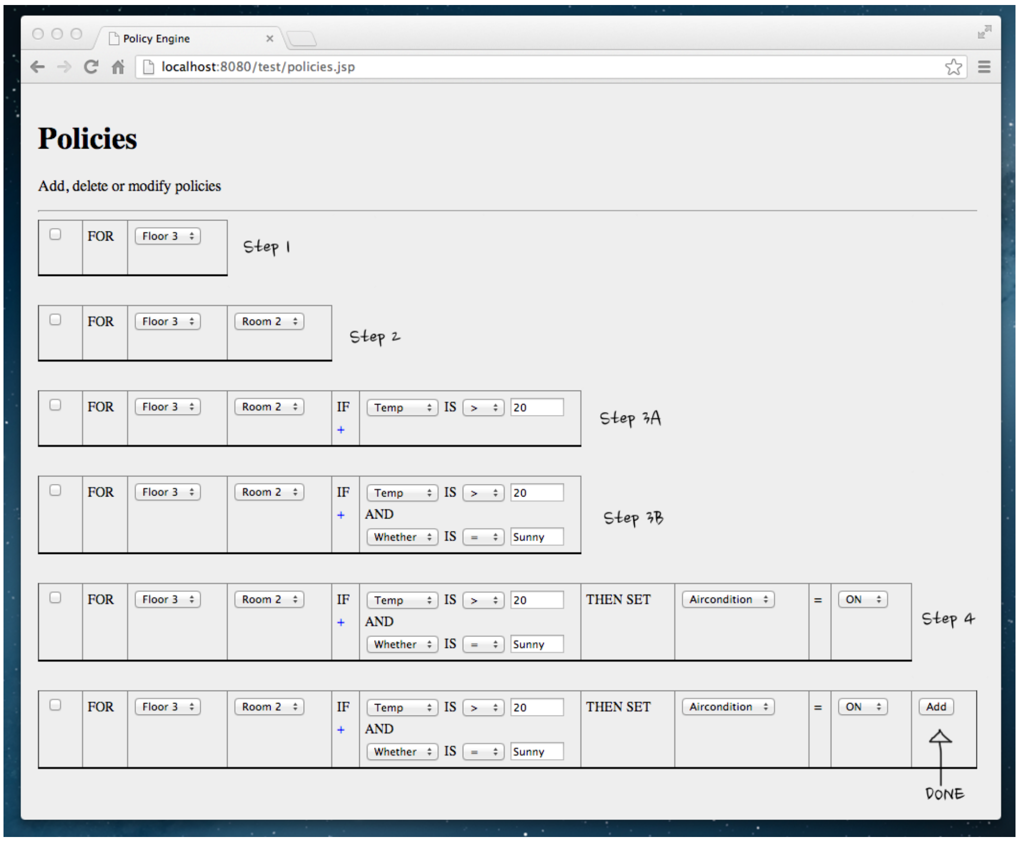
[1]Front-end pages are dynamic HTML website.

5.1.2 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"[17], 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 sites we have thrived to do so.

As an example, we have designed the process of adding new policies, as stepwise selections, instead of presenting the users with all the capabilities and elements at once. The task then looks simpler, and makes the flow of adding a new policy, more easy, and more direct.



Policies

Add, delete or modify policies

☐	FOR	Floor 3								
☐	FOR	Floor 3	Room 2							
☐	FOR	Floor 3	Room 2	IF +	Temp IS > 20					
☐	FOR	Floor 3	Room 2	IF +	Temp IS > 20 AND Whether IS = Sunny					
☐	FOR	Floor 3	Room 2	IF +	Temp IS > 20 AND Whether IS = Sunny	THEN SET	Aircondition	=	ON	
☐	FOR	Floor 3	Room 2	IF +	Temp IS > 20 AND Whether IS = Sunny	THEN SET	Aircondition	=	ON	Add

Figure 5.3: Illustrating the different elements introduced step-by-step. When the user has selected the first elements in "Step 1", then the elements in "Step 2" appears, and so it continues until a final "Add" button completes the process.

Another approach towards usability was using web elements like: Selectors and Buttons, instead of having textual policies, and thereby forcing users to write policy, using the keyboard.

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. Also it forces users to memorize the inputs, and makes controlling the policy engine less intuitive. Therefore the idea was discarded and visual elements was implemented instead.

As for 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.

Chapter 6

Evaluation

In our project we have two areas that we have been evaluating. The first being if our policy engine is actually working and enforcing the defined policies. The other being the front end interface, testing to uncover potential usability issues that needed solving.

6.0.3 Policy Engine System Evaluation

6.0.3.0.13 Log Testing Log every action and check timestamps, values, actions etc...

6.0.3.0.14 JUnit Testing ....

6.0.4 Usability Test

6.0.4.0.15 Think Aloud Test In a thinking aloud test, you ask test participants to use the system while continuously thinking out loud — that is, simply verbalizing their thoughts as they move through the user interface.

To run a basic thinking aloud usability study, you need to do only 3 things:

Recruit representative users. Give them representative tasks to perform. Shut up and let the users do the talking.

Cheap, simple, flexible....

Research shows that testing 5 persons uncover 80% of all usability problems...

REFERENCE "Thinking aloud may be the single most valuable usability engineering method." 1993, Usability Engineering, Jacob Nielsen

6.0.5 Technology

In the beginning of the project we talked about the many technologies available for web development. Obviously there are many approaches to the same solution. Focussing on working as a team, that all have different background experiences we decided to individually write down our development qualifications.

From all of the qualifications in the group, we joined together and made a top three technology approached (see Tabel 6.1. Every team member voted 1, 2 and 3 on an development approach. 1 being the most preferred, and three the less preferred.

Table 6.1: Team members preferred development approaches

Dev environment	Kasper	Thomas	Stefan	Rasmus	Nicolas
PHP, JavaScript, MySQL	2	1	3	3	1
C#.Net, MS SQL	3	2	2	1	1
Java, JSP, Tomcat	1	3	1	2	3

One of the challenges in this project was technology readiness. Every team member has a preferred development language, which ultimately led to a long debate on settling with a development language, and what kind of framework the group should use. The problem was that: either way - some team members would have a harder time participating with implementation than others.

Also some members had already lots of experience in development web applications, while other had very little. Not everyone was familiar with MVC based development, which for those unfamiliar became a challenge. The overall problem was simply having different level of knowledge regarding web development and technologies.

Ultimately however the group decided to go with Java as the main programming language, along with Javascript for front-end parts.

Various milestone for the project was planned and the different implementation tasks, was assigned to different team members. Most noticeable the team unintentionally created subgroups: divided in team members working with front-end tasks, and team members working with backend tasks. However, frequent meetings and online conversations led to positive awareness on what everyone was working with.

Furthermore the team utilize an online project management site, from which the team members could keep each other informed of work in progress.

Also Github was used to easily share and version code and documentation between team members. This made it possible for everyone to see changes and follow ongoing development.

Chapter 7

Discussion

In this section we are going to discuss our project in three different parts, one being the collaboration with the team members from Kenya, the other being the product and lastly the overall project. We have chosen to analyse each of these areas with an approach of highlighting what could have been done differently and perhaps in a better and more successful way. We will reflect on our choices made during the project and finally also on our learning outcomes.

7.0.6 Collaboration

The collaboration between the students at ITU, Denmark and Strathmore, Kenya, proved to be a rather challenging task. Only one person from Strathmore showed interest in the project, even though this should have been four persons. We believe that this is not caused by our approach towards the students from Kenya but instead by a lack of interest and willingness to work on the project. With that being said, there could have been multiple ways we could have achieved a better performing team. Our collaboration strategy is still in the early phases due to the fact that we as a group never got to create any real collaboration platform to work from. A next approach, that we have tried multiple times to do, could be to gather every team member for an online Skype meeting. This way we would be able to, during a discussion of the current project, to develop multiple different subgroups, in the team, and not just the two general subgroups as Denmark and Kenya. With these subgroups in place, one would have a much better foundation and a more solid team.

7.0.7 Product

(It is difficult to discuss the product when it is not done) In the section product, we want to discuss our final product. Questions one could ask:

- What could be improved?
- Prototype?
- Future work?

7.0.8 Project

(It is difficult to discuss the overall project when we still have one month left) A general overview of the project:

- What was good?
- What was not so good?
- How would we approach collaboration in a new project? Do anything different?

Chapter 8

Conclusion

conclusion...

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] 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.

[6] Catherine Durnell Cramton and Pamela J Hinds. Subgroup dynamics in internationally distributed teams: Ethnocentrism or cross-national learning? Research in organizational behavior, 26:231–263, 2004.

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

[8] Cristina B Gibson and Mary E Zellmer-Bruhn. Metaphors and meaning: An intercultural analysis of the concept of teamwork. Administrative Science Quarterly, 46(2):274–303, 2001.

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

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

[11] Jonathan Grudin. Groupware and social dynamics: eight challenges for developers. Communications of the ACM, 37(1):92–105, 1994.

[12] W.B. Gudykunst. Cultural variability in communication. Communication Research, 24(4):327–348, 1997.

[13] 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.

[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] Peter Johnson-Lenz and Trudy Johnson-Lenz. Post-mechanistic groupware primitives: rhythms, boundaries and containers. International Journal of Man-Machine Studies, 34(3):395–417, 1991.

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

[18] 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.

[19] 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.

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

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

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

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

[24] Kjeld Schmidt and Liam Bannon. Taking cscw seriously. Computer Supported Cooperative Work (CSCW), 1(1-2):7–40, 1992.

[25] 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.

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

[27] Marcie J Tyre and Wanda J Orlikowski. Windows of opportunity: Temporal patterns of technological adaptation in organizations. Organization Science, 5(1):98–118, 1994.

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

[29] 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 A.1. Requirements

A.2 A.2. Tests