



university of  
 groningen

faculty of mathematics  
 and natural sciences

# Web-based interface for domain manipulation in Smart Offices

Bachelor Computing Science

July 11, 2013

Student: M. Hoekstra

Primary supervisor: Prof. Dr. Ir. M. Aiello

Secondary supervisor: Prof. Dr. Ir. P. Avgeriou

Daily supervisor: I. Georgievski

# Table of Contents

<b>List of Figures</b> . . . . .	<b>3</b>
<b>1 Introduction</b> . . . . .	<b>4</b>
1.1 Organisation . . . . .	5
1.2 Running example . . . . .	5
<b>2 Smart Environments</b> . . . . .	<b>6</b>
2.1 Planner . . . . .	6
2.2 Hierarchical Planning Definition Language . . . . .	7
2.3 Domain editor . . . . .	8
2.4 Related work . . . . .	9
2.4.1 GreenerBuildings . . . . .	9
2.4.2 Energy Smart Offices . . . . .	9
2.4.3 Task selector . . . . .	10
2.4.4 User interfaces . . . . .	10
<b>3 Requirements and Techniques</b> . . . . .	<b>11</b>
3.1 Requirements . . . . .	11
3.1.1 Domain verification . . . . .	11
3.1.2 User guidance . . . . .	11
3.1.3 Responsiveness . . . . .	11
3.1.4 Organisation . . . . .	11
3.2 Framework . . . . .	11
3.2.1 Play Framework . . . . .	12
3.2.2 Derby Web Design Response . . . . .	12
3.2.3 Twitter Bootstrap . . . . .	12
<b>4 Implementation</b> . . . . .	<b>14</b>
4.1 Architecture . . . . .	14
4.1.1 Model . . . . .	14
4.1.2 View . . . . .	15
4.1.3 Controller . . . . .	15
4.1.4 UML diagram . . . . .	15
4.2 User Guidance . . . . .	16
4.3 Responsiveness . . . . .	17
4.4 Application . . . . .	17
4.4.1 Homepage . . . . .	17
4.4.2 Domain Editor . . . . .	18
4.4.3 Upload . . . . .	20
4.5 Issues . . . . .	21
4.5.1 Saved field values . . . . .	22
4.5.2 No SheepIt! . . . . .	22
4.5.3 Renumbering . . . . .	22
4.5.4 Ambiguity . . . . .	22
4.5.5 Removing . . . . .	23
<b>5 Future work</b> . . . . .	<b>24</b>
5.1 Domain manipulating . . . . .	24
5.1.1 Edit domain . . . . .	24
5.1.2 View domain . . . . .	24
5.1.3 Create domain . . . . .	24

5.2	User authentication . . . . .	24
5.3	Integration . . . . .	25
5.4	Miscellaneous . . . . .	25
<b>6</b>	<b>Conclusion . . . . .</b>	<b>26</b>
	<b>References . . . . .</b>	<b>27</b>
	<b>List of Abbreviations . . . . .</b>	<b>28</b>
	<b>Appendices . . . . .</b>	<b>29</b>
	<b>Appendix A Meeting room . . . . .</b>	<b>29</b>
	<b>Appendix B Structure of a domain and a problem . . . . .</b>	<b>31</b>
	B.1 Domain Description . . . . .	31
	B.2 Problem Description . . . . .	33
	B.3 Requirements . . . . .	34
	<b>Appendix C Installation Guide . . . . .</b>	<b>35</b>

## List of Figures

2.1	GreenerBuildings architecture [11]	9
3.1	Response with different screen sizes [16]	12
3.2	Twitter Bootstrap is used for the design of the form	13
3.3	Pop-overs give more information about a certain part of the form	13
4.1	Collaboration of the MVC components [18]	14
4.2	UML diagram of the application [19]	16
4.3	A text box for entering the name of a variable	17
4.4	The homepage of the application	18
4.5	The form for creating and editing a domain without user input	19
4.6	The form for creating and editing a domain with an error	19
4.7	The part of the form for inserting an action	19
4.8	The part of the form for inserting an action with the field “Atomic Formula” added	20
4.9	The page of the application where users can upload domains	20
4.10	The page of the application where users can upload domains with a parse error	21
4.11	The homepage of the application with a newly uploaded domain	21

# 1 Introduction

Environments such as houses and offices can be made smarter. Where lights had to be turned on and off manually before, this can now be done automatically. For example, when a couple goes on a holiday and they think they have forgotten to turn off the lights or the stove, they can check this on a web application and turn those devices off. They can even set an option to automatically turn off such devices, when they are not in a certain range of those devices. Such environments are called Smart Environments [1,2] and their objectives are to be energy efficient, user-friendly and able to operate devices in a Smart Environment.

An environment possibly consists of many actuators, sensors and users. An *actuator* is a device which affects an environment. A *sensor* is a device which detects input like light or movement. Environments are referred to when speaking of domains. A *domain* is a description of the actuators and actions that can be performed on the actuators in an environment.

Smart Environments can detect, through the available sensors, where persons or actuators are placed in an environment. This allows for different operations on actuators, depending on their placement or even the time. For example, the coffee machine could be turned on when a sensor detects a person waking up. Smart Environments are able to do most operations on the actuators in an environment, so the users do not have to. Although a lot of operations can be done automatically, the user should also have an option to interact with the environment. In order to be able to operate the devices in a Smart Environment, all the users, actuators and sensors are linked to the Internet. A web application provides an interface which allows users to communicate with the Smart Environment.

This web application should be intuitive and user-friendly. Users should be able to operate actuators with the least amount of operations and let the Smart Environment take care of the rest. A user could, for example, desire to turn on a certain light. The user should not have to specify which light has to be turned on or when to turn it off. The Smart Environment can decide which light is meant by the user by sensing the user's location and the nearest light, and when to turn it off by sensing the user's proximity to that light.

In order for the application to be able to operate devices in a Smart Environment, it needs knowledge about an environment. This knowledge has to be modelled by users of a Smart Environment into a domain. This modelling can only be done by users who know which actuators and actions on actuators are available in an environment. Domains have to follow a certain structure which may or may not be known to the users.

The purpose of this work is to design a user-friendly, web-based interface where users can create, view, edit, upload and remove domains. This needs to be done in a way that users do not need to know anything about the structure of a domain or the language the domain is written in. The web-based interface has to be clear for the user and domains have to be represented in an intuitive way. The following questions arise from this problem and have to be answered in order to fulfill the requirements:

1. How can the application allow users to manipulate domains, without them knowing the structure of a domain?
2. How can the application be designed in a user-friendly way and what exactly is needed for it to be user-friendly?
3. How can the application make sure the manipulation of the domains is done in a way that the resulting domain is not faulty with respect to the structure of a domain?
4. How can the access to the application be restricted to the right users and do all the users need to manipulate domains?
5. Does answering all the above questions make sure the application meets the requirements?

## 1.1 Organisation

The content of this work is organised as follows: Section 2 describes Smart Environments and the components which are needed in order to make sure a Smart Environment is energy efficient and user-friendly. Section 3 offers an overview of the techniques used in this work and requirements for this work. Only requirements that are implemented are described here. The architecture along with an overview of the application and the issues that arose during the implementation are handled in section 4. In section 5 the features that have to be implemented are described. This section also provides a description on how to implement those features, along with problems that could arise by implementing those features. And finally, a conclusion about this work and a summary of the answers to the research questions is given in section 6.

## 1.2 Running example

Smart Offices are referred to during this work when talking about Smart Environments, because the application will be used in Smart Offices. A Smart Office is a Smart Environment which is optimised for the use in an office. The difference between a Smart Office and a Smart Environment in a home (called Smart Home) is, for example, that a Smart Office usually has more users and is only used during working hours. The amount and sort of devices in a Smart Office is also different from a Smart Home. And where energy in Smart Offices can be saved in the weekend by turning devices off, energy in Smart Homes can be saved during working hours by turning devices off. Section 2 describes points which distinguish Smart Offices from other types of Smart Environments. The following example is used in this work:

Suppose there is a task called “meeting\_room”. A user executes this task when he or she wants to organise a meeting. The task consists of three actions:

- “turn\_light\_off”: Turns off the lights
- “close\_blinds”: Closes the blinds
- “turn\_beamer\_on”: Turns on the beamer

The three actions are *atomic*, which means that the actions can not be subdivided into other actions. Such actions are also called *primitive* actions.

## 2 Smart Environments

Mark Weiser describes a Smart Environment as “a physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network” [3]. Humans and computers are constantly interacting with each other in Smart Environments. The computers sense the humans’ presence and execute tasks set by humans. In order for the environment to be user-friendly, the computers have to be embedded seamlessly in the environment. Next to being user-friendly, a Smart Environment needs to be energy efficient [4]. In order to be energy efficient, certain actuators in a Smart Environment can be turned off when they are not needed.

A Smart Office is a kind of Smart Environment which applies to working places and differs from other Smart Environments in the following points [2].

- Work in offices can often be automated and is often computer related. For many applications software is already used. Therefore the Smart Office can, besides operating lights, also operate those applications.
- A room can serve different purposes. A room can be a meeting room or can be used as a working place. This means that Smart Offices have to be flexible and thus have to adjust rooms to serve the right purpose.
- Office users are often users who are accustomed to dealing with computers. Therefore, those users could want to configure their working place. Additionally, experienced programmers would not want to be limited by a simple menu when configuring their working place. Different abstraction levels are needed where experienced programmers can write code and where other users can use a menu.
- Smart Offices must be able to deal with multiple users and must be able to support their wishes. Where one user might prefer oral interaction, other users might prefer a web interface to communicate with the system.

In order for the computers to be embedded seamlessly in a Smart Office, a system is needed which is able to interact with users and can make decisions so users do not have to. AI planning [5] is an example of such a system and can be implemented by a planner. This planner somehow has to have knowledge about the environment to be able to make decisions and operate actuators. This knowledge has to be modelled by the users of a Smart Office. All these different components are needed to create a Smart Office and are described in this section.

### 2.1 Planner

A planner is a system which has the goal to solve a so called planning problem. A *planning problem* is the problem of deciding on the actions needed to reach a certain goal from a certain initial state. An *initial state* is the state of the actuators and sensors before a task is executed. Actions are defined as the operations on actuators. Actions can be directly executed by the planner and do not exist of other actions. Tasks are defined as the goals that have to be reached. In order to perform a task, the planner needs to decide on the actions needed to reach the goal. Without tasks, all of the actions in a task have to be executed one by one.

In the running example we have a meeting room and the task is “meeting\_room”. The planner receives the task which consists of the goal that has to be reached, along with an initial state. The planning problem is the decision on which set of actions are needed to reach the goal from the initial state. The goal is to have a room which is prepared for a meeting. An initial state for a light could for example be that the light is turned on. After receiving a goal, the planner will autonomously decide which actions have to be executed in order to fulfill the goal. These actions are the three actions given in the running example. If the task is successfully executed by the planner, the initial state of the room will become the state of the room when a meeting is taking place. For example, the beamer that was turned off in the initial state will be turned on in the final state.

The planner uses the domain to get knowledge about the environment. This knowledge consists of actuators and possible actions that can be performed on those actuators. Knowledge about the initial state of the environment and the goals that have to be reached, is described in a problem. The planning problem in the example above is such a problem. To summarise, users send problems to the planner and the planner uses domains to solve these problems.

The planner that the final application will use is a Hierarchical Task Network (HTN) planner [6]. An HTN planner autonomously chooses a set of actions in order to perform a task. The set of actions that is chosen is called a *plan*. A plan can be made by decomposing tasks into smaller tasks until executable tasks are reached. Executable tasks, which are tasks that do not contain other tasks, are called primitive tasks. Opposite of primitive tasks we have compound tasks which have to be decomposed by the planner. An example of such a primitive task is “turn\_light\_off” which has only one action, namely turning off a certain light.

The planner is useful for creating user-friendly Smart Offices. The user only has to specify a goal and not how the goal can be reached, because the planner will decide on that. Another system that can be used instead of a planner, is a system which executes precomputed actions for every actuator. This however means that for every task, its set of actions has to be predicted for every actuator and every state. This would not be the right solution for Smart Offices [7] where every user has their own expectations and actuators change over time. For example, a disabled person has different needs and expectations from a Smart Office than a person who is not disabled. A disabled person thus needs a different set of actions on a certain actuator. The second issue is that actuators constantly evolve because functions are added or removed. An actuator could also have a different set of actions depending on where it is placed or depending on the time of the day. Without the use of a planner, all the actions have to be predicted which is time consuming and not user-friendly. If however a planner is used and a functionality is added to a certain actuator, the only thing that needs to be done, is to add this functionality to the description of a domain. The planner can autonomously decide when this functionality has to be used.

## 2.2 Hierarchical Planning Definition Language

Hierarchical Planning Definition Language (HPDL) [8] is a language based on the Planning Domain Definition Language (PDDL) [9] and the language described in [10]. HPDL is the language used in the planner, described in section 2.1 and is the language where domains and problems are written in. There are two structures defined in HPDL. One structure allows the description of domains and the other structure allows the description of problems. Thus a planning problem can be solved by combining those two structures. There can be many problems for one domain, all defining their own goals, for example one problem for turning off a light and another problem for turning on a beamer.

The example in listing 1 represents the action “turn\_light\_off”. It has one parameter called “?light”, which is the light that has to be turned off. “state-on” is a predicate which is used for checking whether a light is turned on or off. There are two rules for this predicate:

1. If there exists a predicate “state-on” with a certain light as a parameter, that light is turned on
2. If there is not such a predicate with that light as a parameter, that light is turned off

The precondition checks if the light is turned on or off. If it is on (rule 1), the effect will be executed, otherwise the action will terminate. The effect negates the predicate “state-on ?light”, which means that this predicate is removed from the environment. Because the predicate is removed from the environment, the light is turned off (rule 2).

Listing 1: action turn\_light\_off

```
1 (:action turn_light_off
2   :parameters (?light)
3   :precondition (state-on ?light)
4   :effect (not (state-on ?light))
5 )
```



Listing 2 contains the relevant structure for the example in listing 1. This structure only contains rules relevant for the example. The actual structure of the domain (described in appendix B) is larger and more complex, but is kept out of this example for simplicity.

Listing 2: Structure for turn\_light\_off

```

1 <action-def> ::= (:action <name>
2               :parameters (<variable>)
3               <action-def body>)
4
5 <action-def body> ::= :precondition <pre>
6                   :effect <effect>
7
8 <pre> ::= <atomic formula (<term>)>
9
10 <effect> ::= (not <atomic formula (<term>)>)
11
12 <atomic formula (t)> ::= (<predicate> t*)
13
14 <term> ::= <variable>
15
16 <predicate> ::= <name>
17
18 <variable> ::= ?<name>
19
20 <name> ::= <letter> <any char>*
21
22 <letter> ::= a..z
23
24 <any char> ::= <letter> | _

```

### 2.3 Domain editor

The function of the domain editor is to allow users to describe their environment in a domain and manipulate those domains. Manipulation of domains can be done in the following ways:

- Edit: Already existing domains can be edited. Fields can be added, edited or removed
- Create: New domains can be created
- Upload: Existing domains can be uploaded to the application
- Remove: Existing domains can be removed

The domain editor will be used when a new environment has to be encoded, or when there are actuators or actions that have to be added to or removed from a domain. Not all the users who will use the application should have access to the domain editor. Only experts should have this access, because the domain does not change often. An expert is a user who has the best knowledge of the environment and is capable of translating this knowledge into a domain description. The domain describes the environment, therefore it must not have errors or miss certain actuators or actions, because all the user's operations depend on the domain. For example, when a domain lacks knowledge about operations on a certain actuator, no operations can be done on that actuator via the application.

If users want to add or modify tasks based on their personal preferences, it should be possible for those users to create a domain which only affects themselves. When such users then modify a domain, these modifications are not visible to other users and therefore mistakes and newly added features affect only the users who have modified the domain.

## 2.4 Related work

The field of Smart Environments is a broad field. Every environment has its own needs and many projects are devoted to create an energy efficient and user-friendly environment. This section describes projects related to this work.

### 2.4.1 GreenerBuildings

GreenerBuildings aims to realise an integrated solution that addresses the challenge of energy-aware adaptation from basic (energy harvesting) sensors and actuators, up to an embedded software for coordinating thousands of smart objects with the goals of energy saving and user support [11].

The purpose of the GreenerBuildings project is to develop an energy-aware aware framework based on embedded service middleware and a building-distributed architecture of smart objects. The framework relies on advances of ubiquitous ultra-low power sensing, sensor-based human activity recognition, and device orchestration, to guarantee responsiveness, scalability, and dependability in its goal to achieve energy savings at the whole building level.

The GreenerBuildings architecture specifically emphasises occupant activity and behaviour as key element for adaptation, but addresses other building context information as well. The building adaptation concept foreseen in GreenerBuildings follows a layered representation to decouple different abstractions as seen in figure 2.1. The ubiquitous system layer consists of physical devices, in particular building-distributed ubiquitous sensing, processing, and actuation architecture of GreenerBuildings. The service composition layer comprises the abstract composition and orchestration functionalities of the energy-aware framework.

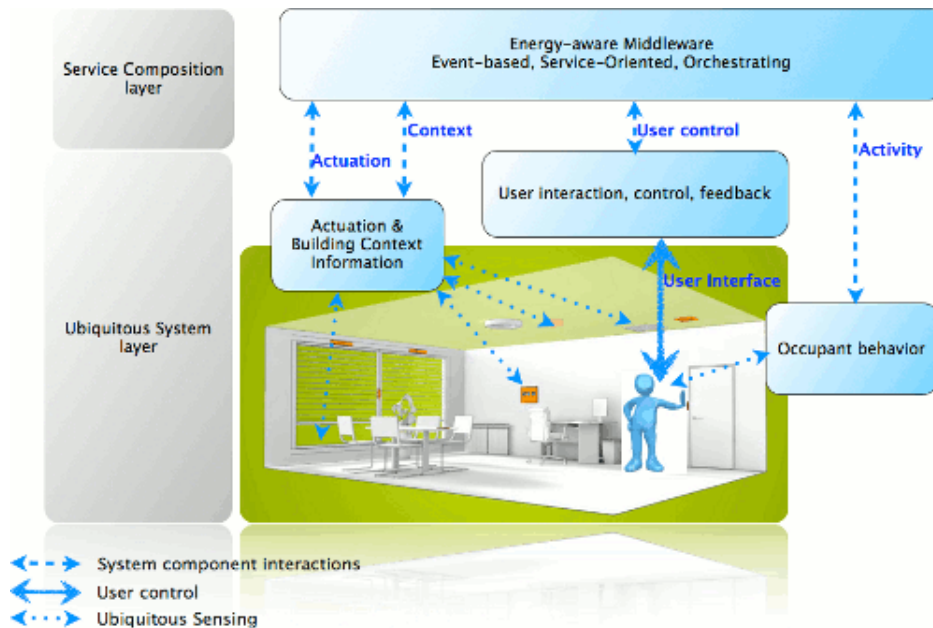


Figure 2.1: GreenerBuildings architecture [11]

### 2.4.2 Energy Smart Offices

The Energy Smart Offices (EnSO) project [12] will develop a scalable energy-aware framework for offices that incorporates awareness of the occupant's and building context with the goal of overall energy saving. The EnSO approach builds on cooperative, goal-oriented sensing, context information processing, activity and plan recognition, service composition and device actuation in large-scale distributed sensor networks to minimise energy consumption while adhering to occupant comfort requirements.

The goal of the EnSO project is to couple advanced research and novel technique in ambient network technology, probabilistic activity and the plan recognition with innovative service-oriented approaches, thus developing a truly energy-aware platform for the offices.

The major innovative aspect of EnSO is the transformation of offices into smart energy-aware and dynamically adaptive systems. This is a radical step ahead from the current state of the art, where control of installation and appliances is either left to the user or based on simple feedback control loops. Instead, offices will be able to detect the context of the user and building and perform a set of actions in order to optimise energy consumption and adhere to user needs (via the coordination and composition of available sensor and software services). The solutions aim to minimise installation and maintenance costs, while allowing retrofitting into existing buildings. Beyond the specific office application, EnSO will also contribute to the study of indoor sensor technology, activity recognition, behaviour understanding, service composition, concurrent planning and embedded middleware solutions.

### 2.4.3 Task selector

In order to complete this work, another part of the application has to be written which is called the task selector [13]. The function of the task selector is to allow users to set goals which have to be reached by the planner. The task selector handles operations on problems. The task “meeting\_room” can be executed in this task selector by a user. A problem consisting of the goals that need to be reached, including the initial state of the environment will be send to the HTN planner and the planner will try to solve a planning problem. The task selector depends on the domain editor, because it needs knowledge about the environment in order to know which tasks can be executed by the user. This knowledge can be extracted from a domain. Unlike the domain editor, which is used sporadically, the task selector is used far more often and has to be accessible to anyone working in an office having the rights to use the application.

### 2.4.4 User interfaces

Human-Computer Interaction (HCI) can be made possible in a lot of different ways, like interacting through a display or with speech. Butz describes among others the following ways [14].

- The most devices used for HCI are displays. Displays can be categorised along several dimensions among which are their size and form factor, their position and orientation in space and their degree of mobility. These dimensions can be manipulated in order to be the most effective in a certain environment or for a certain application. A smaller display will probably be used in multiple environments because it is designed to be mobile.
- Interactive Display Surfaces combine visual output with input on the same surface. A tablet or an interactive whiteboard are examples of such an interface. Those surfaces support multiple contact points which allows the surface to be used by multiple users at the same time.
- Tangible/Physical interface is an interface which provides physical objects for the user to grasp to operate a device. The advantage of this interface is that it is physical and humans have learned to manipulate physical objects all their life.

## 3 Requirements and Techniques

In this section the requirements for this work are described, along with the techniques that are used to implement those requirements.

### 3.1 Requirements

An overview of the requirements for this project is given in this section. This overview is a combination of the features the application needs to offer and the requirements which specify how the application has to look like. Requirements not yet implemented are discussed in section 5.

#### 3.1.1 Domain verification

The application should be able to parse uploaded domains, to check if the uploaded file corresponds to the structure of a domain. When the parser stumbles upon errors, a message is reported to the user indicating the line number of the error and a user-friendly error message. Domains which contain errors should not be accepted by the application to avoid problems that the task selector can have while reading the domain. Faulty domains lead to faulty problems, which lead to the planner not being able to handle the problems and therefore users are not able to execute tasks.

#### 3.1.2 User guidance

The domain editor allows users to edit or create domains. Changes in domains do not have to be parsed, because the form for creating or editing domains only accepts the domain if the form is error free. In fact, the form for domain editing should be more restrictive than the structure for a domain. HPDL could be adjusted to allow only logically valid operations, but this would keep HPDL from being simple and general. The form has to guide the user in choosing the right options, which also ensures that the user does not need understanding of the underlying language. This aspect is important for the application to be user-friendly. This guidance of the user is further described in section 4.2.

#### 3.1.3 Responsiveness

In a Smart Office there are a lot of users all preferring different kinds of devices to interact with the system. Therefore the application has to be suitable for all kinds of devices and screen sizes.

#### 3.1.4 Organisation

A domain can be infinitely large, therefore the form for editing and creating domains should be organised. Different frameworks have to be used to keep the application and the code clear. The application could use pop-overs for more information on certain fields. It should be possible to hide fields to keep the page from becoming too large so the user does not need to scroll a lot.

### 3.2 Framework

To develop better and faster, the application uses frameworks. Using frameworks allows the programmer to focus more on the implementation of features (after the framework is understood) and less on the code that is needed to develop those features on. Also, with most frameworks, the programmers know that the framework's behaviour is correct and that it is well optimised. The project will be organised and is easier to understand for future developers working on the project. A framework keeps the programmer from writing the same code over and over because modules can be reused. To provide a user-friendly, intuitive design, multiple front-end frameworks are used. Derby Web Design Response and Twitter bootstrap are used to make the application appealing and easy to use for the user. The Play Framework is used as a back-end framework, for it can satisfy all the requirements. The frameworks used are described in this section.

### 3.2.1 Play Framework

The back-end framework the application is written in, is the Play Framework [15]. Besides HTML, JavaScript and JQuery, Scala and Java can be used (interchangeably) for writing a function. Java is the main language of the project and Scala, among other uses, is used in parsing. Play Framework offers an organised overview of errors if there are any and changes are loaded into the application by simply refreshing the website.

### 3.2.2 Derby Web Design Response

A user-friendly, simple and clear web design, is Response from Derby Web Design (DWD) [16]. It is free to use, under the condition that DWD will be credited in the footer on the website. Response is responsive which means that the interface will be adjusted to the screen size. Therefore the web application looks different on a PC, tablet or mobile as seen in figure 3.1.

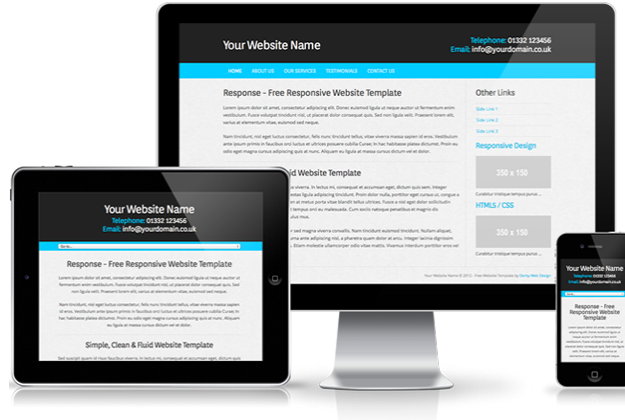


Figure 3.1: Response with different screen sizes [16]

### 3.2.3 Twitter Bootstrap

Users need to enter a lot of information when creating or editing domains. To provide a clear form, the Twitter Bootstrap framework design [17] is used. Bootstrap not only provides a neat design, it also provides a lot of reusable components and JQuery plugins. It is for example possible to hide and show certain fields to keep the form clear and pop-overs provide extra information to the user. Buttons with different purposes have different colours and placement in the form, to keep it simple and easy to understand for the user. Figure 3.2 shows an example of a such a form and figure 3.3 shows an example of a pop-over.

The screenshot shows a web form titled "New Domain". At the top, there is a navigation bar with "Home > New Domain" and links for "DOMAINS", "NEW DOMAIN", and "UPLOAD DOMAIN". Below the navigation bar, the main heading is "Create a new domain". The form is organized into several sections: "Requirements", "Types", "Predicates", "Actions", and "Tasks". The "Predicates" section is currently active and contains a form for "Predicate 1". This form has a "Name" input field with the placeholder "Name of the Predicate", a "Variable" section with an "Add variable" button, and an "Add predicate" button. A "Remove" button is located in the top right corner of the "Predicate 1" section. At the bottom of the form, there are "Insert" and "Cancel" buttons. A small copyright notice "Mark Hoekstra © 2013 - Free Website Template by Derby Web Design" is visible at the bottom right of the page.

Figure 3.2: Twitter Bootstrap is used for the design of the form

This image shows a close-up of a form section titled "Methods". There is a green "Add parameter" button and a "Methods" label. A white pop-over tooltip is displayed over the "Methods" label, containing the text: "Add methods. The minimum amount of methods is one."

Figure 3.3: Pop-overs give more information about a certain part of the form

## 4 Implementation

The implementation of the application should be flexible and easy to understand, because new features have to be added and other developers will work with it. Comments are added in lines of code which are not easy to understand. Also the use of frameworks makes it easier for future developers working with the application. The usage of frameworks is easy to learn and well documented. This section does not describe the implementation of the complete domain editor because not all of the features are implemented. Features that still have to be implemented are described in section 5.

### 4.1 Architecture

The application is based on the Model-View-Controller (MVC) design pattern, which ensures that the application remains clear, as shown in figure 4.1. The Play Framework uses this model and also has another component called “public” which is common in web developing. The component public consists of the stylesheets, Javascript files and images used in the application. The separation of those components ensures that the application is well organised and easy to understand.

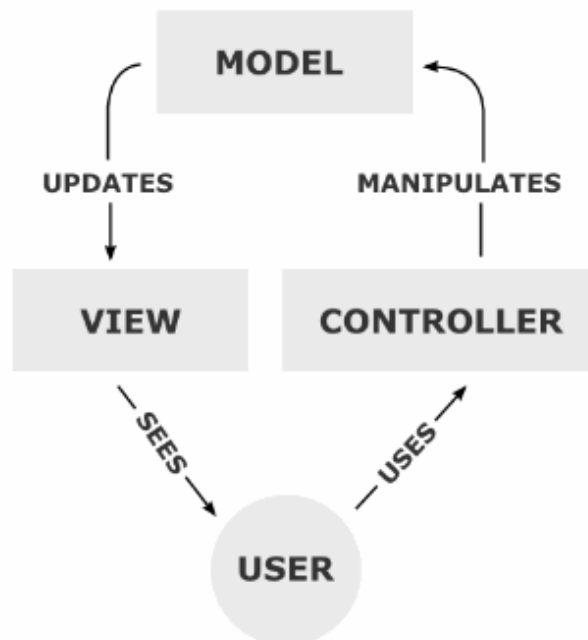


Figure 4.1: Collaboration of the MVC components [18]

#### 4.1.1 Model

In MVC, the model consists of the data used in an application. In the domain editor, this corresponds to the Java object file which contains knowledge about a domain. The plain text domain files, stored in the application, are also part of the model. As is common with storing files for web applications, the files are stored on the server rather than in a database. If in the future it is needed to link users to a file, the path name of the file should be stored in a database.

### 4.1.2 View

The view consists of all the web pages of the application, in other words what the user sees. The Play Framework makes it possible to have a web page which is a combination of HTML and Scala code. The Javascript code is put in a separate file from the web pages to keep it clear. Another advantage of this, is that when a Javascript file or a stylesheet is modified, the page reloads instantly. This comes from the fact that the code of those files is executed on the client side and thus on the machine of the user. When the web page itself is modified the code needs to be compiled first, because this code is executed on the server side. This compiling takes time so, from a developers point of view, it is recommended to separate the client side and the server side code.

### 4.1.3 Controller

The controller consists of the code which is executed when a user performs an action. When a user for example clicks on the upload button the controller takes over. The controller will do the necessary checks like parsing and checking if the name is unique and will redirect the user to the home page if there are no errors.

### 4.1.4 UML diagram

The Unified Modeling Language (UML) diagram in figure 4.2, gives an overview of the most important Java classes in the application. Less important classes and methods of classes are not shown to keep the overview clear. The connection between the New and Edit classes indicates that the Edit class refers to the New class once the Edit class has read a file into a Domain class. The New and Edit classes have a connection with the Domain class because the Domain class is created in those classes.

The New class has the objective to perform error checking on the submitted form for domain creation/editing and will write the domain to a file once it is free from errors. The Edit class reads a domain from a file into the Domain class and redirects to the form for editing. The class called Application is used on the home page. It retrieves domains from the server to show them in a list and it allows for the removing and downloading of domains. The job of the Upload class is to parse uploaded files and to store those files on the server if they are free from errors. Lastly, the Domain class contains the knowledge of a domain, such as actions and tasks.



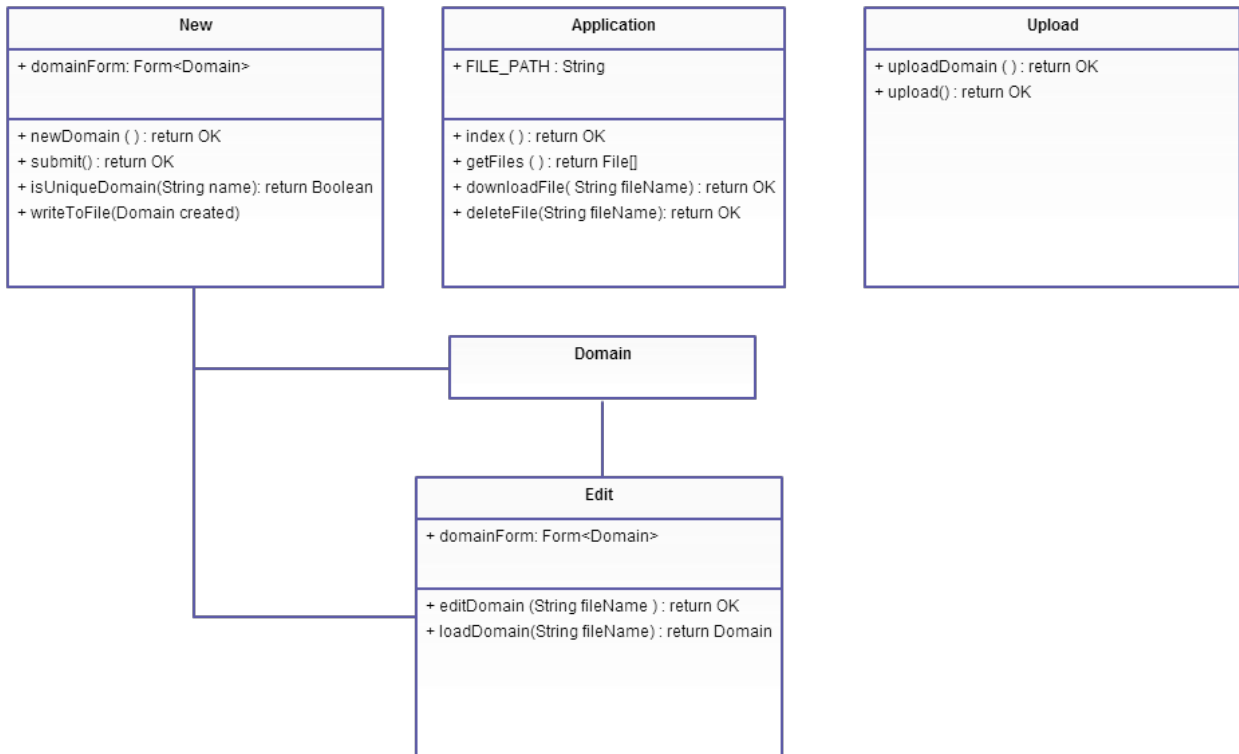


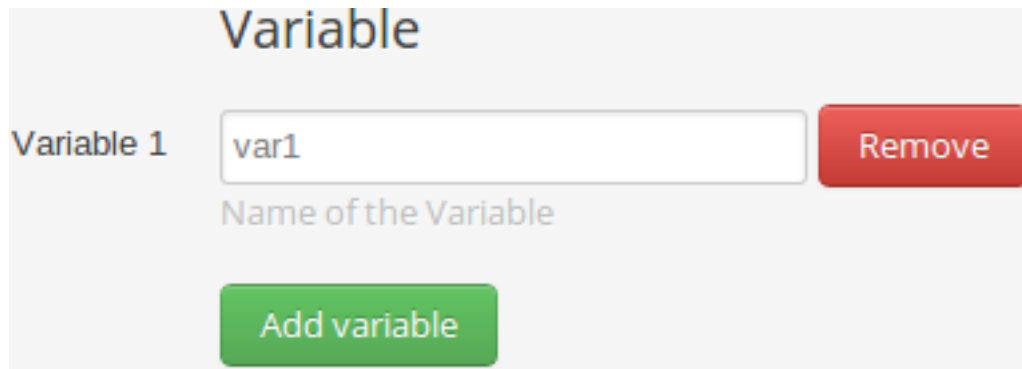
Figure 4.2: UML diagram of the application [19]

## 4.2 User Guidance

To make sure the application lives up to the requirement of user-friendliness, the user will be guided through the form for domain manipulating. This guidance results in that the user does not need to have an understanding of HPDL in order to use the application. To guide the user, possible logically valid but practically unnecessary operations are not allowed in the form and certain details are hidden from the user. Firstly, an example of unnecessary operations is described, and secondly, an example of hidden details is described.

An example of an unnecessary operation is the double negation which becomes positive again. Suppose the user has chosen the option “not” in a certain part of the form by clicking on a button called NOT. “not” is defined as the negation of the expression that follows. Choosing “not” for the second time is of no use, because it negates the negation which makes the expression positive. Therefore when the user clicks on the button NOT, this button will become invisible. Note that when the user removes the option “not”, the button NOT will become visible again.

An example of hidden details, is certain parts of the structure of a domain. Every domain has to start with a bracket followed by the keyword “define”. This keyword is of no use to the user and it will make the form less clear, therefore this keyword is not shown in the form. Another example is the definition of a variable. A variable has to start with a question mark directly followed by its name. This name has some constraints on it but that is not relevant for this example. The only thing the user sees, is a text box as shown in figure 4.3, with a label on the side saying this text box is a variable. The question mark is hidden to the user and the user does not have to insert it. The question mark is placed before the name of the variable hidden from the view.



The screenshot shows a form titled "Variable". On the left, it says "Variable 1". To its right is a text input field containing the text "var1". Further right is a red button labeled "Remove". Below the input field, the text "Name of the Variable" is displayed in a lighter color. At the bottom center of the form is a green button labeled "Add variable".

Figure 4.3: A text box for entering the name of a variable

Important actions like removing a domain or removing multiple fields, are actions that can not easily be reversed. To make sure that those important actions are really what the user wants to do, and not a mistake, confirmation messages appear to minimise such mistakes. A confirmation message appears for example when the user wants to remove multiple fields, which is described in section 4.5.5.

Error detection is also part of the user guidance. The application should not allow users to create domains which contain errors. Therefore the user will be notified if there are any errors at the place where the error is found, with a message describing the error.

### 4.3 Responsiveness

The application is not totally responsive, but this should not be a major issue because the application is only used a few times for an environment. Also, editing domains will mostly be done on a desktop PC (where it is optimised for) because an environment usually has a lot of actions and actuators, which all have to be modelled into a domain.

### 4.4 Application

This section gives an overview of what the application looks like using screenshots.

#### 4.4.1 Homepage

The homepage of the application consists of a table containing the available domains as seen in figure 4.4. From this page the domains can be edited, viewed, downloaded and removed. Clicking on “edit” will take the user to the domain editor, which is described in section 4.4.2.

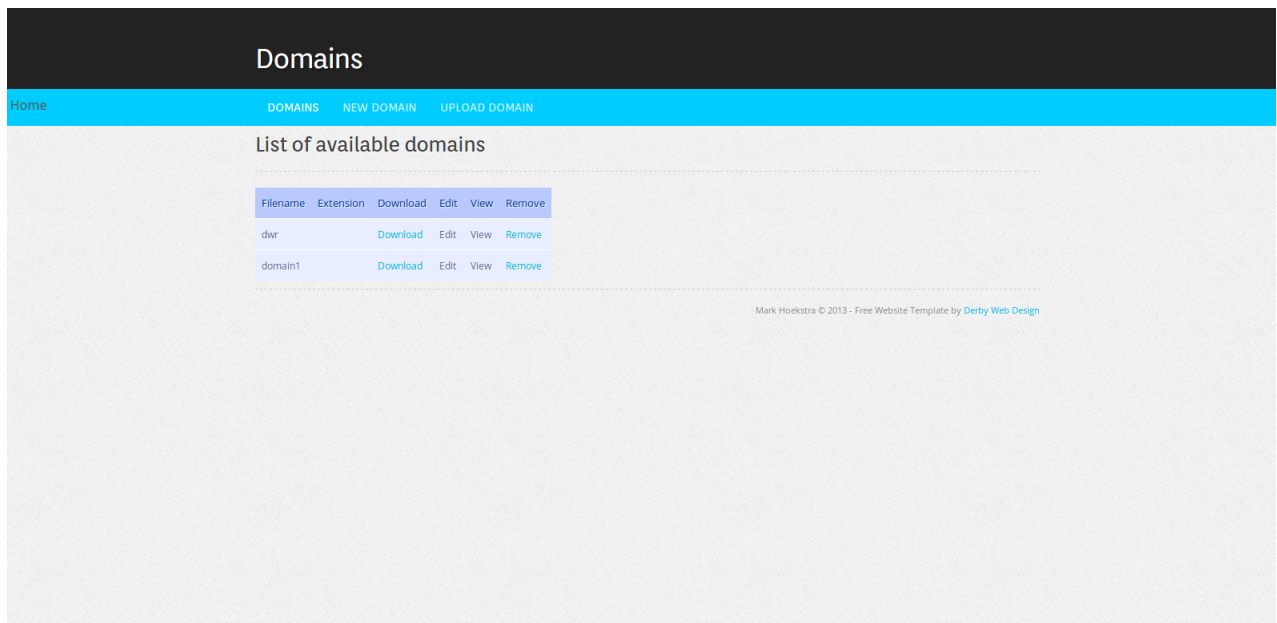


Figure 4.4: The homepage of the application

#### 4.4.2 Domain Editor

This page contains a form where users can add and delete fields in order to create a description of the environment. Initially the form will look like figure 4.5. An example of a domain which can be created with the domain editor is given in appendix A.

The screenshot shows a web interface titled "New Domain". At the top, there is a navigation bar with "Home > New Domain" and "DOMAINS NEW DOMAIN UPLOAD DOMAIN". Below this is a section titled "Create a new domain". It features a "Domain name" input field with a placeholder "Name of the Domain". Below the input field are five stacked, light-colored boxes labeled "Requirements", "Types", "Predicates", "Actions", and "Tasks". At the bottom of the form are "Insert" and "Cancel" buttons. A small copyright notice "Mark Hoekstra © 2013 - Free Website Template by Derby Web Design" is visible at the bottom right.

Figure 4.5: The form for creating and editing a domain without user input

Figure 4.6 shows an example of where the user has done something wrong. In this case the user has entered a name for a domain which is not correct.

This screenshot shows the "Create a new domain" form with an error. A red banner at the top says "Please fix all errors". Below it, the "Domain name" input field contains the character "1". A red border surrounds the input field, and a red error message to its right reads: "The name has to start with a letter followed by a letter, digit, '-' or '\_'". The placeholder text "Name of the Domain" is visible below the input field.

Figure 4.6: The form for creating and editing a domain with an error

An example of an action is showed in figure 4.7.

The screenshot displays the "Action 1" form. It has a "Name" input field with a placeholder "Name of the Action" and a "Remove" button in the top right corner. Below the name field are three sections: "Parameter" with an "Add parameter" button; "Precondition" with buttons for "Atomic formula", "And", "Or", "Not", "Imply", "For all", and "f\_comp"; and "Effect" with buttons for "For all", "Atomic formula", and "Assign".

Figure 4.7: The part of the form for inserting an action

Figure 4.8 shows an example of user guidance. Where in figure 4.7 the buttons for adding a precondition were visible, they are not visible in figure 4.8. The structure of a domain only allows one “Atomic Formula” to be added in this case. Hiding the precondition buttons in this stage will keep the user from building forth on that error.

Figure 4.8: The part of the form for inserting an action with the field “Atomic Formula” added

#### 4.4.3 Upload

This page allows users to upload existing domains as seen in figure 4.9. When the domain is submitted, a parser will check the domain on errors. The application will then take the user to the homepage if the domain is parsed successfully and will stay on the page when there are errors.

Figure 4.9: The page of the application where users can upload domains

In figure 4.10 the user has uploaded a domain with the missing keyword “define”. This results in a parse error.

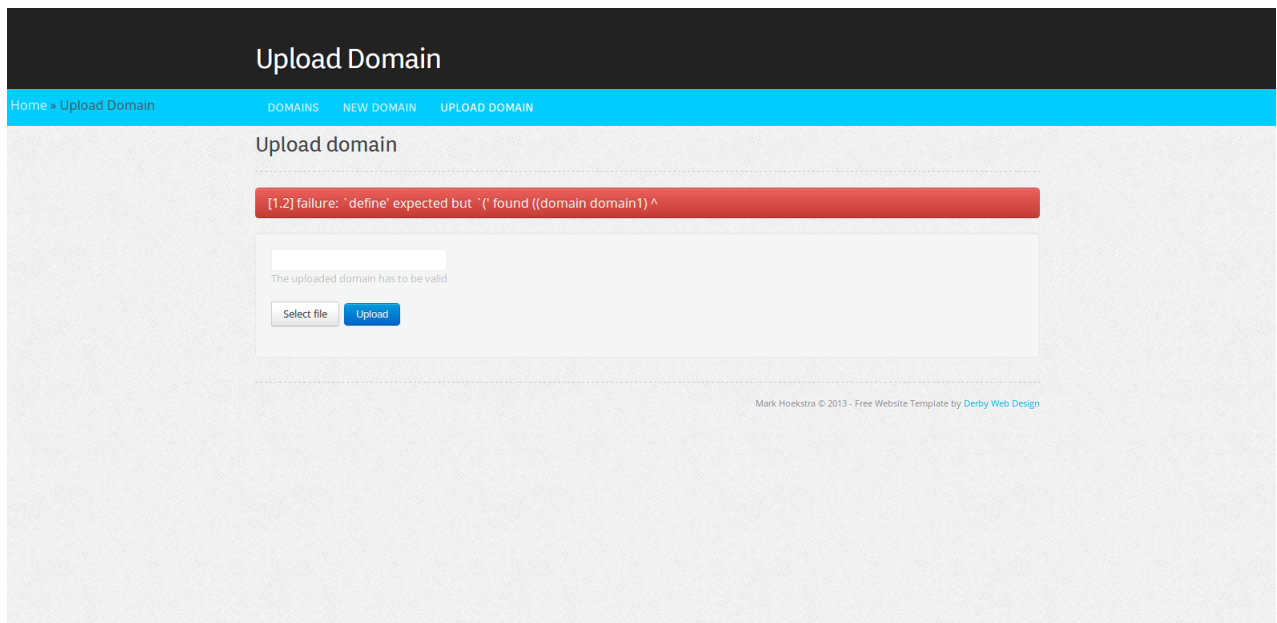


Figure 4.10: The page of the application where users can upload domains with a parse error

In figure 4.11 the user has uploaded a domain which is parsed successfully.

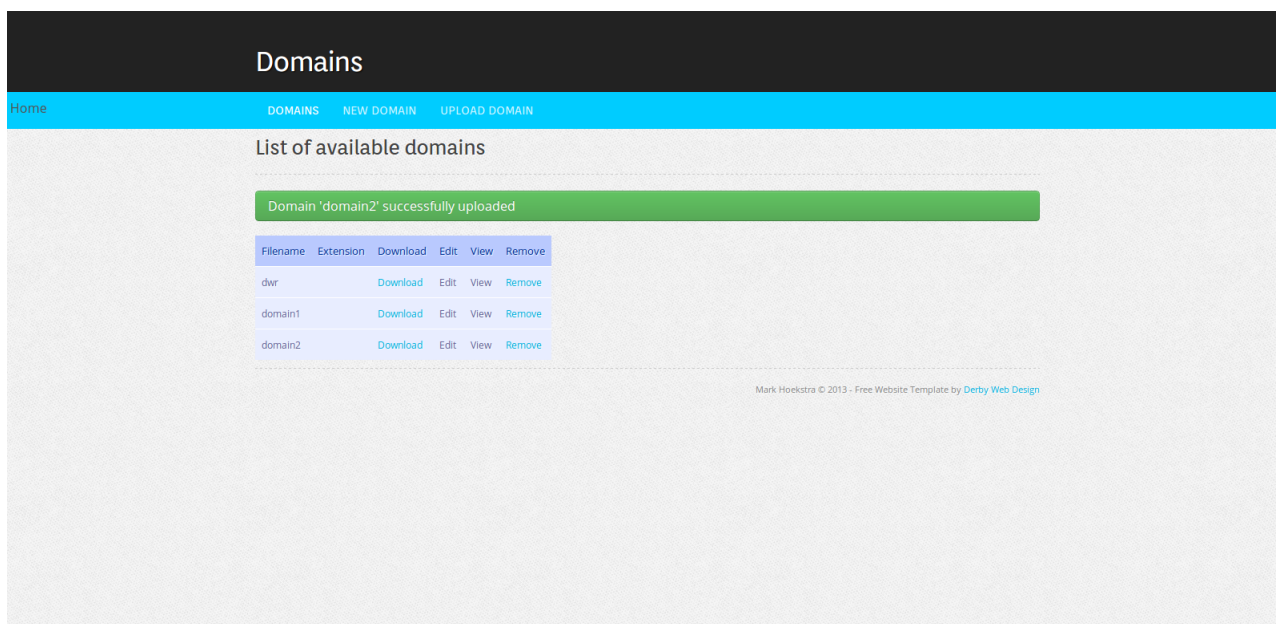


Figure 4.11: The homepage of the application with a newly uploaded domain

## 4.5 Issues

During the implementation of the application, some problems arose. In this section the largest problems and the solutions for them are discussed.

### 4.5.1 Saved field values

When a user uses a form and enters some values, he or she will eventually submit the form. On submitting the form, the Play Framework will automatically check if the entered values correspond to the constraints given by the programmer. When there are errors, the form will not be accepted and the user has to be notified of the errors so they can be fixed. To fulfill the requirement of user-friendliness, the previous entered values should not get lost. The Play Framework will handle this and the programmer only has to specify that the fields have to be repeated when there are errors. This method works well on non-dynamic forms, which are forms which do not change in size. The domains can be of various sizes so the form has to be dynamic. Users can also choose between different options in a certain field. For example, the user can decide to add a variable or a name at some point in the form. Thus a field could be a variable or a number. This means that when the user submits a form, which has errors, the order and entered values of the previous fields have to be kept intact. The solution to this problem is to add a type to a field, which specifies what kind of field it is. When the application then repeats previously entered fields, it has to check the type of the field and perform the necessary action on it.

### 4.5.2 No SheepIt!

SheepIt! [20] is a JQuery plug-in which allows easy field cloning. In the domain editor the user should have the option to add and delete fields. The SheepIt! plug-in provides this option with a few extra features like removing all the fields, add a specific number of fields at the same time and remove only the last field. To use the plug-in it had to be combined with the Play Framework. This plug-in is discarded because the best it could do, was to add and remove fields dynamically, but not repeat the previously entered fields for a form with errors. The solution for this problem was to implement the adding and removing of fields without the use of the plug-in.

### 4.5.3 Renumbering

The form used in the domain editor has to have an object in which it can store information about a domain. Error checking can then be performed on the object and it can finally be written to a file. Because a domain does not have a fixed size, some mechanism has to be invented to differ specific fields of the same type. Every field which can have multiple entries is stored in a list. To separate items in the list, an index is used. Therefore, when adding an item in a field, it has to get a unique number. The application uses a renumbering procedure which renumbers the items in those list when items are added or removed. This way, the data of the form can be combined to the object for the domain and every field is unique.

### 4.5.4 Ambiguity

It is possible to have a precondition for an action. This means that the action is only executed when this precondition is true. This condition can exist of an infinite amount of logical expressions. The user can for example specify that “condition1” and “condition2” must hold before the action can be executed. The domain structure allows different operations to bind those conditions like “and” or “or”. It is also possible to have an “and” nested in an “or” or vice versa. “and” and “or” can both have an infinite amount of logical expressions in them. HPDL uses parentheses to bind the expressions to the right operators, to make sure there is no ambiguity. This means that there also must be a way for the user to specify those parentheses. There must not be an option “insert bracket” because the user may not know what this means and with brackets you can quickly lose the overview. The solution is to allow the user to choose the option “end” which tells that the expressions before the “end” are part of the operator before them. The option “end” is only visible when there is an operator that can be ended. When the user does not insert “end”, the application will assume that the end of the operator is after the last expression and will automatically place the right amount of parentheses.

### 4.5.5 Removing

Users must have the option to remove certain fields, because it is not user-friendly to let the user remake the whole domain if he or she made an error. Removing a field will disorganise the numbering of the fields, as described in section 4.5.3. The solution to this problem is to renumber all the affected fields. Another problem that arises, is the removing of a field which has nested fields in it, as described in section 4.5.4. The solution to this problem is to remove the field including all the fields nested in it and then renumber all the affected fields. Removing multiple fields requires confirmation from the user to confirm that the user really wants to remove multiple fields. Removing a field can not always be done, because sometimes removing a field is not allowed. For example, when a field has to have a minimum of one field and an open operator is ended after this field. This scenario occurs when the sequence of fields is: “and”, “not”, “X”, “end” where “X” can be any one field. Removing “X” will result in having a field “not” followed by “end”. “not” is a field which has to have a minimum of one field (“end” does not count as a field) therefore removing “X” is not allowed in this case.



## 5 Future work

This section describes the features that are not implemented yet, along with possible solutions and problems which can arise when implementing them.

### 5.1 Domain manipulating

The manipulation of domains is not fully implemented. The following points still need to be implemented.

#### 5.1.1 Edit domain

The possibility of editing already existing domains. To make this possible, a domain file has to be read into a Java object file. Once this object file is created, the Play Framework offers the possibility to open a pre-filled form. This means that the code for creating a domain is the same as the code for editing a domain. So when a user wants to edit a domain, it will be redirected to the form used for creating a domain, where the user finds a pre-filled form. From this point, fields can be edited, added or removed. For now, the editing only supports requirements, predicates, parameters of actions and most of the fields from a precondition in an action.

#### 5.1.2 View domain

The possibility of viewing a domain. This should be done in a user-friendly way. So there must be a possibility of hiding fields, to avoid having a large web page just as in the creating or editing of a domain. Also, one must keep in mind that the user may not have a full understanding of HPDL, so the viewing of a domain has to be simple and easy to understand. For example, parentheses do not have to be visible, for it will make the page more complicated. In the form for creating or editing a domain there are no brackets, and nested fields are made visible by putting a border around the field as shown in figure 4.8. The possibility of viewing a domain helps the user to learn about the actuators and actions on those actuators in an environment. The user should be able to have options to only show specific parts of a domain. Examples of such options are to only show all the actions or to only show all the predicates of a domain.

#### 5.1.3 Create domain

The creation of a domain is not entirely finished. The domain can specify which requirements have to be used. These requirements influence the options in the form. For example “:negative-preconditions” is a requirement that allows the use of “not” (negation) in the domain. When this requirement is not chosen, the form has to hide the buttons for “not”.

The form is not complete with regard to error checking. It is for example still possible to create a domain which does not follow the structure. The form should check on all the errors prior to inserting the domain, without the need of parsing.

### 5.2 User authentication

The application should ask users to login/sign up. Only users who have the rights should be allowed to access the application. A user’s e-mail address could for example be cross-referenced with the database of a company to know if an employee has the rights to access the application. There have to be two types of users:

- Normal users
- Expert users

Normal users should only have access to the task selector part of the application, where expert users have access to the task selector and the domain editor. A database should also be added to the application which can securely store the user information. The user information will at least exist of a user name, an e-mail address and a password, but could be extended with their personal domains or other settings. The e-mail address can then be used as a unique id and it can be combined with the database of a company to check if the user has to have access to a specific region in the application.

When user profiles are implemented, there should be some kind of a system to make sure only one domain can be edited at a time and that a domain can not be removed when someone is editing it. The latter can easily be prevented by checking if someone is using a domain that another user wants to remove. The user wanting to remove the domain will get a message saying that the domain is currently in use and that it is currently not possible to remove the domain. The former can possibly be solved in the following ways:

- Duplicate domain: The first user that is done editing can insert the domain without restrictions. The domain of the next user will be renamed and the user will receive a message saying his or her domain is renamed because someone else edited it.
- Ostrich algorithm: Do nothing and let the last user overwrite the domain. Because there is a restricted access on the domain editor it could be quite rare that users are editing the same domain at the same time. This is probably not the best way, but it is a possible solution.
- Block: When a user edits a domain he or she will be the only one possible of editing that domain. Other users wanting to edit that domain will receive a message saying that the domain is currently in use. This blocking should not be for an unlimited amount of time. It could for example happen that the application thinks the user is editing the domain while there is a power failure.

### 5.3 Integration

In order for this application to be fully working, it has to be combined with the task selector and the HTN planner. The domain editor and the task selector are both written in the Play Framework and use the same front-end frameworks to make the integration easier. The domain editor, the task selector and the HTN planner all should work as one system where the planner will be used by the task selector and where the task selector and the planner depend on the domains which can be manipulated in the domain editor.

### 5.4 Miscellaneous

There are other features that could be implemented which are summed up here:

- A button to add multiple fields at once in the creation or editing of a domain
- The option to show and hide certain fields
- More explanatory text to inform the user when he or she creates or edits a domain
- The error messages from the parser should be more informative

## 6 Conclusion

The aim of this work is to create a user-friendly, web-based interface for the manipulation of domains. This interface will be used in Smart Offices, to allow users to describe their environment in a domain. The application is universal enough to be used in other Smart Environments that make use of an HTN planning system. Before creating such an application, questions had to be answered in this work which are summarised in this section.

1. How can the application allow users to manipulate domains, without them knowing the structure of a domain?

The application has to guide the user into choosing the right options and has to give informative messages to the user when the user makes mistakes. The application must only show the important parts of a domain and not the details such as parentheses or keywords. Uploaded files are first verified before they are accepted, to make sure all the domains on the server contain no errors.

2. How can the application be designed in a user-friendly way and what exactly is needed for it to be user-friendly?

Front-end frameworks are a great way of providing a well structured and user-friendly design which is optimised to the needs of the user. In order for the application to be user-friendly, there should be a minimum of confirmation messages and the user should not need to learn how to use the application.

3. How can the application make sure the manipulation of the domains is done in a way that the resulting domain is not faulty with respect to the structure of a domain?

The user is guided during the manipulation of a domain, which keeps the user from adding fields which are not allowed and the entered values are checked on errors. A domain is only accepted if the user has submitted the form without errors.

4. How can the access to the application be restricted to the right users and do all the users need to manipulate domains?

The e-mail addresses from users wanting to create an account should be cross-referenced with the database of a company, in order to decide if the user has the rights. Not all the users should be allowed to manipulate domains. Only users which we call experts are allowed to manipulate domains. It should however be possible for normal users to create a domain based on their personal preferences which does not affect other users.

5. Does answering all the above questions make sure the application meets the requirements?

Questions 1 and 3 address the requirement that the application has to be able to guide the user in creating or editing a domain and that domains should contain no errors. The requirement of user-friendliness is covered by questions 2. Answering question 4 will allow the application to support multiple users.

Currently users of the application are able to manipulate the most part of the domain in a user-friendly manner. A domain is represented in an intuitive way, where certain details are hidden from the user. Uploaded, edited and created domains are checked on most errors. In conclusion, there are still some features which have not been implemented yet, but, given more time, they can be added to the application.

## References

- [1] J. C. Augusto, H. Nakashima, and H. Aghajan, “Ambient intelligence and smart environments: A state of the art,” in *in Handbook of Ambient Intelligence and Smart Environments*, pp. 3–31, Springer, 2010.
- [2] D. J. Cook and S. K. Das, *Smart Environments Technology, Protocols, and Applications*. New Jersey: Wiley-Interscience, 1 edition ed., 2005.
- [3] M. Weiser, R. Gold, and J. S. Brown, “The origins of ubiquitous computing research at parc in the late 1980s,” *IBM Syst. J.*, vol. 38, pp. 693–696, Dec. 1999.
- [4] I. Georgievski, D. Degeler, G. A. Pagani, T. A. Nguyen, A. Lazovik, and M. Aiello, “Optimizing energy costs for offices connected to the smart grid,” *IEEE Transactions on Smart Grid*, 2012.
- [5] J. Hendler, A. Tate, and M. Drummond, “Ai planning: systems and techniques,” tech. rep., College Park, MD, USA, 1990.
- [6] I. Georgievski and M. Aiello, “An overview of hierarchical task network planning,” tech. rep., University of Groningen, JBI 2012-12-5, 2012.
- [7] E. Kaldeli, E. U. Warriach, A. Lazovik, and M. Aiello, “Coordinating the web of services for a smart home,” *ACM Trans. Web*, vol. 7, pp. 10:1–10:40, May 2013.
- [8] I. Georgievski, “Hierarchical planning definition language,” tech. rep., University of Groningen, JBI 2013-12-3, 2013.
- [9] A. Gerevini and D. Long, “Plan constraints and preferences in pddl3 - the language of the fifth international planning competition,” tech. rep., 2005.
- [10] L. Castillo, J. Fdez-olivares, Óscar García-pérez, and F. Palao, “Efficiently handling temporal knowledge in an htn planner,” in *In Sixteenth International Conference on Automated Planning and Scheduling, ICAPS*, pp. 63–72, AAAI, 2006.
- [11] “Greenerbuildings project.” <http://www.greenerbuildings.eu/>. Accessed: 2013-07-03.
- [12] “Enso project.” <http://www.ensoffices.nl/>. Accessed: 2013-07-03.
- [13] J. de Boer, “Smart offices: A web-based interface for task selection.” Bachelor thesis, Rijksuniversiteit Groningen, 2013.
- [14] A. Butz, “User interfaces and HCI for ambient intelligence and smart environments,” in *Handbook of Ambient Intelligence and Smart Environments*, pp. 535–558, 2010.
- [15] “Play framework.” <http://www.playframework.com/>. version 2.0.4.
- [16] “Derby web design response.” <http://www.derby-webdesign.co.uk/>.
- [17] “Twitter bootstrap.” <http://twitter.github.io/bootstrap/>.
- [18] Wikipedia, “Mvc.” <http://en.wikipedia.org/wiki/Model-view-controller>, 2013. Accessed: 2013-07-04.
- [19] “Created with.” <http://www.creately.com/>.
- [20] M. D. Rosso, “Sheepit!” [http://www.mdelrosso.com/sheepit/?lng=en\\_GB](http://www.mdelrosso.com/sheepit/?lng=en_GB).

## List of Abbreviations

DWD	Derby Web Design
EnSO	Energy Smart Offices
HCI	Human-Computer Interaction
HPDL	Hierarchical Planning Definition Language
HTML	HyperText Markup Language
HTN	Hierarchical Task Network
MVC	Model-View-Controller
PDDL	Planning Domain Definition Language
UML	Unified Modeling Language

# Appendices

## Appendix A Meeting room

Listing 3 contains the code for a domain in a Smart Office written in HPDL called “smartoffices”. This domain contains knowledge about tasks and actions that can be performed in a room used for meetings. The tasks that can be performed are “set-projector”, “set-blinds” and “set-meeting-room”. Every task will execute actions in order to reach a certain goal if the precondition is true.

Listing 3: Domain in a Smart Office

```
1 (define (domain smartoffices)
2 (:requirements :strips :typing :negative-preconditions :universal-preconditions :conditional
   -effects)
3 (:types floor room
4       light screen
5       projectionScreen projector
6       blindsheight blindsangle
7       integer
8 )
9 (:predicates
10  (in ?floor ?room)
11  (in ?room ?something)
12  (turned-on ?device)
13  (light-value ?light ?luxLevel)
14  (light-value-sensor ?sensor ?luxLevel)
15 )
16
17 (:action turn-on-screen
18  :parameters (?f - floor ?r - room ?s - screen)
19  :precondition (and (in ?r ?f) (in ?s ?r) (not (turned-on ?s)))
20  :effect (turned-on ?s)
21 )
22
23 (:action turn-on-all-lights
24  :parameters (?f - floor ?r - room)
25  :precondition (and (in ?r ?f) (forall (?l - light) (in ?l ?r) (not (turned-on ?l))))
26  :effect (forall (?l - light) (in ?l ?r) (turned-on ?l))
27 )
28
29 (:action turn-off-screen
30  :parameters (?f - floor ?r - room ?s - screen)
31  :precondition (and (in ?r ?f) (in ?s ?r) (turned-on ?s))
32  :effect (not (turned-on ?s))
33 )
34
35 (:action dim-light
36  :parameters (?f - floor ?r - room ?l - light ?ll)
37  :precondition (and (in ?r ?f) (in ?l ?r))
38  :effect (light-value ?l ?ll)
39 )
40
41 (:action dim-lights
42  :parameters (?f - floor ?r - room ?ll - integer)
43  :precondition (and (in ?r ?f) (forall (?l - light) (in ?l ?r) ()))
44  :effect (forall (?l - light) (in ?l ?r) (light-value ?l ?ll))
45 )
46
47 (:action turn-on-projector
48  :parameters (?f - floor ?r - room ?p - projector)
49  :precondition (and (in ?r ?f) (in ?p ?r) (not (turned-on ?p)))
50  :effect (turned-on ?p)
51 )
```

```

52
53 (:action turn-off-projector
54   :parameters (?f - floor ?r - room ?p - projector)
55   :precondition (and (in ?r ?f) (in ?p ?r) (turned-on ?p))
56   :effect (not (turned-on ?p))
57 )
58
59 (:action set-projection-screen
60   :parameters (?f - floor ?r - room ?ps - projectionScreen)
61   :precondition (and (in ?r ?f) (in ?ps ?r) (not (turned-on ?ps)))
62   :effect (turned-on ?ps)
63 )
64
65 (:action set-blinds-height
66   :parameters (?f - floor ?r - room ?b - blindsheight ?h - integer)
67   :precondition (and (in ?r ?f) (in ?b ?r) (blindsheight-value ?b ?bh))
68   :effect (and (not (blindsheight-value ?b ?bh)) (blindsheight-value ?b ?h))
69 )
70
71 (:action set-blinds-angle
72   :parameters (?f - floor ?r - room ?b - blindsangle ?a - integer)
73   :precondition (and (in ?r ?f) (in ?b ?r) (blindsangle-value ?b ?ba))
74   :effect (and (not (blindsangle-value ?b ?ba)) (blindsangle-value ?b ?a))
75 )
76
77 (:task set-projector
78   :parameters (?f - floor ?r - room)
79   (:method with-projection-screen
80     :precondition (and (in ?r ?f) (projectionScreen ?ps) (projector ?p) (in ?ps ?r) (in ?p
81       ?r) (not (turned-on ?ps)) (not (turned-on ?p))))
82     :tasks (sequence (set-projection-screen ?f ?r ?ps) (turn-on-projector ?f ?r ?p))
83   )
84   (:method with-projection-screen
85     :precondition (and (in ?r ?f) (projectionScreen ?ps) (projector ?p) (in ?ps ?r) (in ?p
86       ?r) (turned-on ?ps) (turned-on ?p))
87     :tasks ()
88   )
89 )
90 (:task set-blinds
91   :parameters (?f - floor ?r - room ?h ?a - integer)
92   (:method only-one-case
93     :precondition (and (in ?r ?f) (blindsheight ?bh) (blindsangle ?ba) (in ?bh ?r) (in ?ba
94       ?r))
95     :tasks (sequence (set-blinds-height ?f ?r ?bh ?h) (set-blinds-angle ?f ?r ?ba ?a))
96   )
97 )
98 (:task set-meeting-room
99   :parameters (?f - floor ?r - room)
100   (:method too-much-natural-light
101     :precondition (and (light ?ll) (light-value-sensor ?ll ?llv) (> ?llv 1000) (in ?r ?f))
102     :tasks (sequence (set-blinds ?f ?r 1000 0) (dim-lights ?f ?r 200) (set-projector ?f ?r
103       ))
104   )
105   (:method enough-light
106     :precondition (and (light ?ll) (light-value-sensor ?ll ?llv) (> ?llv 200) (< ?llv
107       1000) (in ?r ?f))
108     :tasks (sequence (dim-lights ?f ?r 200) (set-projector ?f ?r))
109   )

```

## Appendix B Structure of a domain and a problem

This appendix contains the description of a domain and a problem. Lines with requirements in superscript in them, mean that the line will only be executed by the planner if the requirement in superscript is contained in the list of requirements of a domain. For example, if the requirement “:typing” is chosen, the planner will execute every line where the requirement is placed in superscript. If “:typing” is not chosen, the planner will ignore those lines.

### B.1 Domain Description

```

<domain> ::= (define (domain <name>)
               <require-def>?
               <types-def>?:typing
               <predicates-def>?
               <structure-def>*)

<require-def> ::= (:requirements <require-key>+)
<require-key> ::= See Section B.3

<types-def> ::= (:types <typed list (<name>)>)
<typed list (x)> ::= x*
<typed list (x)> :=:typing x+ - <type> <typed list (x)>
<type> ::= <name>
<type> ::= object

<predicates-def> ::= (:predicates <atomic formula skeleton>+)
<atomic formula skeleton> ::= (<predicate> <typed list (<variable>)>)
<predicate> ::= <name>
<variable> ::= ?<name>

<structure-def> ::= <action-def>
<structure-def> ::= <task-def>
<structure-def> :=:derived-predicates <derived-def>

<emptyOr (x)> ::= ()
<emptyOr (x)> ::= x

<action-def> ::= (:action <name>
                  :parameters (<typed list (<variable>)>))
                  <action-def body>)
<action-def body> ::= :precondition <emptyOr (<pre>)>
                  :effect <emptyOr (<effect>)>

<pre> ::= <atomic formula (<term>)>
<pre> :=:negative-preconditions <literal (<term>)>
<pre> ::= (and <pre>*)
<pre> :=:disjunctive-preconditions (or <pre>*)
<pre> :=:disjunctive-preconditions (not <pre>)
<pre> :=:disjunctive-preconditions (imply <pre> <pre>)
<pre> :=:universal-preconditions (forall (<typed list (<variable>)>)) <pre>
<pre> :=:numeric-fluents <f-comp>

```



```

<atomic formula (t)> ::= (<predicate> t*)
<literal (t)> ::= <atomic formula (t)>
<literal (t)> ::= (not <atomic formula (t)>)
<term> ::= <name>
<term> ::= <variable>
<term> ::= <number>

<f-comp> ::= (<binary-comp> <f-exp> <f-exp>)
<binary-comp> ::= >
<binary-comp> ::= <
<binary-comp> ::= =
<binary-comp> ::= <=
<binary-comp> ::= >=
<f-exp> ::=:numeric-fluents <number>
<f-exp> ::=:numeric-fluents (<binary-op> <f-exp> <f-exp>)
<f-exp> ::=:numeric-fluents (<multi-op> <f-exp> <f-exp>+)
<f-exp> ::=:numeric-fluents (- <f-exp>)
<f-exp> ::=:numeric-fluents <f-head>
<binary-op> ::= <multi-op>
<binary-op> ::= -
<binary-op> ::= /
<multi-op> ::= *
<multi-op> ::= +
<f-head> ::= (<function-symbol> <term>*)
<f-head> ::= <function-symbol>
<function-symbol> ::= <name>

<effect> ::= (and <c-effect>*)
<effect> ::= <c-effect>
<c-effect> ::=:conditional-effects (forall (<typed list (<variable>))) <effect>)
<c-effect> ::= <p-effect>
<p-effect> ::= (not <atomic formula (<term>)>)
<p-effect> ::= <atomic formula (<term>)>
<p-effect> ::=:numeric-fluents (<assign-op> <f-head> <f-exp>)
<assign-op> ::= assign
<assign-op> ::= scale-up
<assign-op> ::= scale-down
<assign-op> ::= increase
<assign-op> ::= decrease

<task-def> ::= (:task <name>
                :parameters (<typed list (<variable>)>)
                <task-def body>)
<task-def body> ::= <method>+
<method> ::= (:method <name>
                :precondition <emptyOr (<pre>)>
                :tasks <emptyOr (<task-list>)>))
<task-list> ::= (<ordering> <task-atom>*)
<ordering> ::= sequence
<ordering> ::= unordered
<task-atom> ::= (<name> <term>*)

```

```
<derived-def>          ::= (:derived <atomic formula skeleton> <pre>)

<name>                 ::= <letter> <any char>*
<letter>               ::= a..z | A..Z
<any char>             ::= <letter>
<any char>             ::= <digit>
<any char>             ::= -
<any char>             ::= _
<number>               ::= <digit>+ <decimal>?
<digit>                ::= 0..9
<decimal>              ::= .<digit>+
```

## B.2 Problem Description

```
<problem>              ::= (define (problem <name>)
                             (:domain <name>)
                             <require-def>?
                             <object-declaration>?<init>
                             <goal-tasks>)

<require-def>          ::= (:requirements <require-key>+)
<require-key>          ::= See Section B.3

<object-declaration>  ::= (:objects <typed list (<name>)>)

<init>                 ::= (:init <init-el>*)
<init-el>              ::= <literal (<name>)>
<atomic formula (t)>   ::= (<predicate> t*)
<literal (t)>          ::= <atomic formula (t)>
<predicate> = <name>

<goal-tasks>          ::= (:goal-tasks <task-list>)
<task-list>           ::= (<ordering> <task-atom>*)
<ordering>            ::= sequence
<ordering>            ::= unordered
<task-atom>           ::= (<name> <term>*)

<name>                 ::= <letter> <any char>*
<letter>               ::= a..z | A..Z
<any char>             ::= <letter>
<any char>             ::= <digit>
<any char>             ::= -
<any char>             ::= _
<number>               ::= <digit>+ <decimal>?
<digit>                ::= 0..9
```

### B.3 Requirements

A table of currently supported requirements by the SH planner. If a domain specifies no requirements, it is assumed a requirement `:strips`.

<code>:strips</code>	Basic STRIPS-style adds and deletes
<code>:typing</code>	Allow types in declarations of variables
<code>:negative-preconditions</code>	Allow <code>not</code> in precondition descriptions
<code>:disjunctive-preconditions</code>	Allow <code>or</code> in precondition descriptions
<code>:universal-preconditions</code>	Allow <code>forall</code> in precondition descriptions
<code>:numeric-fluents</code>	Allow use of effects using assignment operators and arithmetic preconditions
<code>:conditional-effects</code>	Allow use of effects with <code>forall</code> operator
<code>:derived-predicates</code>	Allows predicates whose truth value is defined by a formula

## Appendix C Installation Guide

The application runs on the Play Framework version 2.04, therefore this version of the Play Framework has to be installed before running the application. Documentation on downloading and installing the Play Framework can be found on [www.playframework.com](http://www.playframework.com). When the framework is installed, the application can be downloaded from <https://github.com/MarkHoekstra/SODomain>. After downloading, the application can be run from the folder where the application is downloaded/extracted to with the following command: *play run*. The application is running and can be found on `localhost:9000`