



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

*Distributed
Computing*



Load-Balancing of Consumers in Electricity Networks

Master's Thesis

Christof Baumann

baumachr@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich



Supervisors:

Dipl. Math. Stephan Holzer (stholzer@tik.ee.ethz.ch)
Miguel Rodriguez (miguel.rodriguez@aizo.com)
Prof. Dr. Roger Wattenhofer (wattenhofer@tik.ee.ethz.ch)

February 29, 2012

Acknowledgements

I would like to thank my mentors Stephan Holzer, Miguel Rodriguez and Roger Wattenhofer, who always had time for me when I needed them.

Many thanks also to all the colleagues at aizo for supporting me and answering many of my questions. Especially, I would like to express my thanks to Andreas Brauchli, who gave me an excellent introduction in digitalSTROM app development and the ExtJS4 framework. I would also like to reward Christian Hitz and Reto Flütsch for always having time for me when questions appeared.

I am grateful to my wife Marianne for helping me on some of the artwork and supporting me during my studies. She encouraged and withstood me in good and bad times of the work process.

Abstract

This thesis describes the implementation and analysis of an automated energy load-balancing system for digitalSTROM enabled households. The idea is that an energy provider gets access to user configured devices to use them to balance the energy consumption load. Further the thesis investigates algorithms to be implemented at the energy provider to balance the overall consumption load using configurable devices in households.

Contents

Acknowledgements	i
Abstract	ii
1 Introduction	1
2 digitalSTROM®	3
2.1 Configuration of a digitalSTROM System	4
2.2 digitalSTROM Server	4
2.3 Electricity Markets	7
2.3.1 Derivatives Market	8
2.3.2 Spot Market	8
2.3.3 Intraday Market	8
3 Smart-Grid App	10
3.1 Idea	10
3.2 Configuration	10
3.2.1 Delayed ON	11
3.2.2 Short OFF	12
3.2.3 Example Configurations	12
3.3 Dataflow	13
3.3.1 Protocol	14
3.4 Property Tree	15
3.5 The Configuration User Interface	16
3.6 Detecting Device Presence	17
3.7 Demonstration: Energy Provider	18
3.8 Deploying the System	18

4 Load-Balancing Algorithms	23
4.1 Simple Reacting Algorithm	23
4.2 2D Packing Algorithm	25
4.2.1 Heuristic to Find the Best Starting Point	26
4.3 Simulating the Algorithms	27
4.3.1 Results	29
5 Device Detection and Consumption Prediction	33
5.1 Detecting Device Consumption Pattern	33
5.2 Consumption and Availability Predictions	35
6 Social Aspects	36
6.1 Sales Appeal	36
6.2 Security	36
6.3 Privacy	37
7 Conclusions & Future Work	39
Bibliography	41
A Implementation Notes	1
A.1 Smart-Grid App	1
A.1.1 Subscriptions to Events	1
A.1.2 Script	1
A.1.3 User Interface	2
A.2 Demonstration Energy Provider	2
A.3 Simulation	3
B Source Code	5
B.1 Smart-Grid App	6
B.1.1 Subscriptions to Events	6
B.1.2 Script	6
B.1.3 User Interface	21
B.2 Demonstration Energy Provider	29

CONTENTS

v

B.3 Simulation	45
--------------------------	----

CHAPTER 1

Introduction

In the future much more of the consumed electricity will be produced as renewable energies like wind or solar energy. Several issues need to be sorted out such as where to take the energy from if there is currently no wind or sun. Or what to do with excess energy in case of too much sun or wind. Currently the energy providers have to either shut down a power plant or start up an additional plant to compensate these imbalances. Those processes can be very expensive and in general they are a waste of energy. Another option the energy providers have in times of energy exuberance is to store the energy in other places. For example the energy can be transformed to potential energy and back to electrical energy later when it is needed. This is done in Switzerland with some barrier lakes. In times of energy excess water is pumped upwards to the lakes and again flushed down if not enough energy is available. This is a very clever storage facility except for the fact that in every transformation there is also loss involved. Additionally only a limited amount of energy can be stored that way.

To address those problems it would be desirable to have a possibility to regulate the consumer side of the network and not only react on the producer side. All energy providers have to regulate the availability of energy according to the consumption. Apart from the ripple control system [9] they don't have an instrument to control the consumption. Ripple control systems are widely spread these days to control the starting times of devices. A ripple control sender sends signals over the power line that are received in the households and may trigger the start of devices. Disadvantages of the ripple control system are that communication is only unidirectional and its expandability is limited. Normally the system is just used during the night to stagger devices that consume large amounts of energy like boilers.

An energy provider predicts the future consumption needs and has to fulfill those needs under any circumstance. If it fails in satisfying the consumption this results in a power outage. The only way to control this is to either start up or shut down power plants or to store energy at another place. With an instrument to regulate the consumer side the set of actions can be enlarged and the overall efficiency of the network can be increased. Of course this regulation

should not have an impact on the comfort of the customer. Instead the system should be able to shift consumption peaks and to store energy in the household where it can be used directly and does not have to be transformed again. The user of such a system should not miss any comfort and should not notice when load-balancing occurs.

There are already some pilot projects running that try to achieve the goal of balancing the consumption. An example of such a project is the pilot study of ienergie¹ in Ittigen Switzerland [3]. They try to animate the consumers to shift their consumptions to periods of larger availability. Except for their product “Flex” they want to find out if users are willing to change their consumption habits by just being informed about the availability and consumption curves. The users should themselves shift their consumption behavior without the help of the system. The product “Flex” provides a way to regulate some special devices, like boilers or heat pumps, over the GSM/UMTS network by an energy provider instead of the established ripple control [9].

In this thesis I would like to generalize the concept of publishing devices to an energy provider that can use those devices to load-balance the energy consumption. I worked together with the company aizo². Aizo is a start up company that is developing and selling the digitalSTROM home automation system³.

In the chapter 2 we describe the digitalSTROM automation system in further detail and try to give some insight into the topic of energy markets. The 3rd chapter covers the implementation, testing and evaluation of the extension application for digitalSTROM that was developed in this thesis. We discuss and evaluate load-balancing algorithms that make use of the user configured devices in the chapter 4. Then in chapter 5 we propose a method to detect consumption patterns of devices by just looking at the overall consumption of the household. Finally in the chapter 6 we try to analyze the proposed load-balancing system from the users point of view.

¹<http://inergie.ch>

²<http://www.aizo.com>

³<http://www.digitalstrom.org>

CHAPTER 2

digitalSTROM®

The specialty of the digitalSTROM automation system as described in [11] is that it communicates over the existing power line. Some of the details about the power line communication are documented in [4]. The system is ideal to be deployed in existing buildings, because no additional cables have to be installed. Just the end points that should be used with digitalSTROM (like light bulbs and light switches) need to have a module installed. A digitalSTROM system normally consist of the following components:

- *digitalSTROM Meter (dSM)*: Several *dSMs* are deployed in the fuse box of a building. There is one needed for every current circuit in the building. It communicates with the clamps in the building over power line and with other *dSMs* over an RS485 bus.
- *digitalSTROM Filter (dSF)*: The *dSF* is mainly used to reduce interference with other devices. It conditions the signals on the power line and does corrections on the 50Hz sine wave. Examples of devices that introduce interference are switching power supplies or solar panels. The latter mainly because of the transformation from direct current to alternating current. There is one *dSF* needed per phase.
- *digitalSTROM Server (dSS)*: The server is connected to the same RS485 bus as all the *dSMs*. It features an Ethernet adapter to connect to the local network or internet. The *dSS* is used to enrich the functionality of the system. An installation would work without a *dSS* but then some features like time triggered events would be missing.
- Clamps are deployed everywhere in the house. Every power plug, every light bulb, every light switch and every device that should be used with digitalSTROM has to be equipped with a clamp. As seen in the figure 2.1 there exist several types of clamps. These are categorized in color groups to do auto configuration. The black joker clamp can be configured to any other color. Every clamp in the system has its unique *digitalSTROM ID*

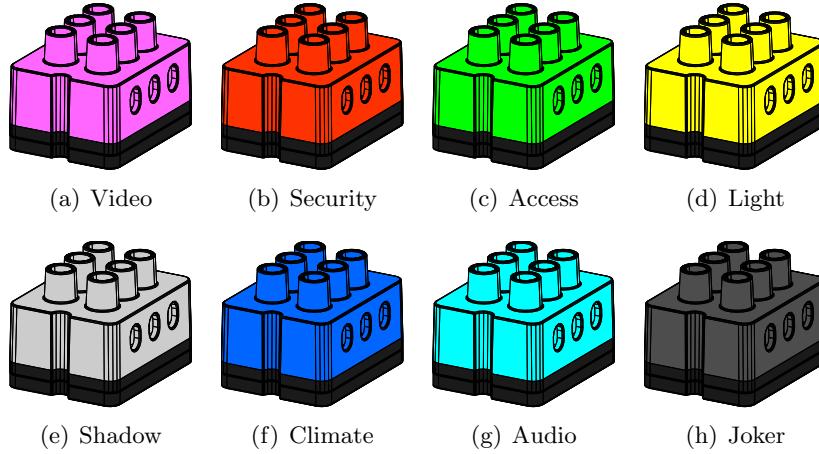


Figure 2.1: Colors of the clamps

(*dSID*), that it used to identify and address the clamp. The functionality provided is different for each clamp type. A light clamp for example contains the hardware to dim and switch loads up to 150W. A blind clamp includes two relays to drive the blind's motors.

An overview of a digitalSTROM installation is sketched in the figure 2.2.

2.1 Configuration of a digitalSTROM System

The system can be configured in two ways:

- Use a normal light switch to configure the system with specialized patterns of clicks.
- Use the web *User Interface (UI)* of the *dSS* that has to be connected to the local network infrastructure. A screenshot of the web *UI* can be seen in the figure 2.3.

For further information about the usage and configuration of the system take a look at the users manual on the digitalSTROM website [1].

2.2 digitalSTROM Server

During my thesis I mainly worked with the *dSS*. The *dSS* is an ARM powered device running an embedded Linux operating system. I wrote an extensions program (app) that can be installed on the *dSS* using the apps page of the

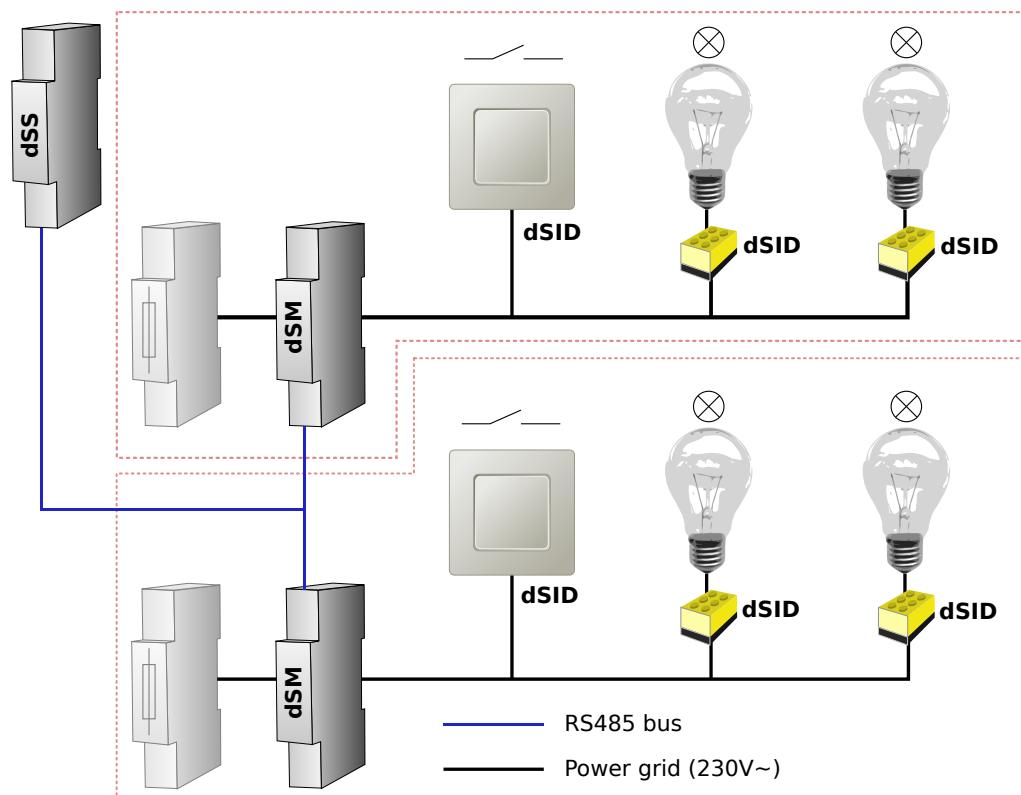


Figure 2.2: Overview of a digitalSTROM installation

digitalSTROM-Konfigurator



Apps SW: Aktivitäten SW: Räume SW: System Hardware Hilfe

Menü

Verwenden
Installieren
Update

Verwenden

App öffnen App deinstallieren

scene-resp... timed-events

1.3.0-test10 | Ansicht aktualisieren Standard Ansicht

(a) Apps

digitalSTROM-Konfigurator



Apps SW: Aktivitäten SW: Räume SW: System **Hardware** Hilfe

Meter

dSM Name	dSM ID	Anzahl Geräte	Verbr...	ARM ...	DSP ...	HW V...
dsm-cbumae0	0000085b	1	0	1.2.1.1	1.1.0.1	11.1....
dsm-cbumae1	00000964	2	0	1.2.1.1	1.1.0.1	11.1....

Geräte

Name	dSID	Ausgang	Taster Mod...	Taster Ein...	Raum	H...	Fir...	Regi...	Zulet...	...
Boiler	000075cc	geschaltet	Gerät		testRaum	1	GE...	3.1.4	L0:26:11	L4:18:28	
Taster	00006338	deaktiviert	Bereich 1, ...		testRaum	1	GE...	3.1.5	L0:50:18	L4:18:28	

1.3.0-test10 | Ansicht aktualisieren Standard Ansicht

(b) Hardware overview

Figure 2.3: Screenshots of the web user interface of the dSS.

configuration web *UI* as seen in figure 2.3(a). An extension app is executed in a sandbox inside the main *dSS* process. An app may contain the following parts:

- Subscriptions to events that are triggered either by the hardware or by the app software. Events that are raised by the software can be triggered with a specifiable delay.
- Scripts written in JavaScript that are interpreted by the *dSS* main process on event raises.
- A web *UI* to do configurations or visualizations.

The *dSS* provides multiple *Application Programming Interfaces (APIs)* to access the functionality of the system. First it provides a *JavaScript Object Notation (JSON)*¹ and a *Simple Object Access Protocol (SOAP)*² *API* that are accessible over a secure HTTP connection. Those *APIs* are the most complete because all of the existing apps heavily rely on them. They are intended for communication between the *UI* of the apps and the *dSS*. The third *API* available is the internal app *JavaScript API*. The app code that runs on the *dSS* upon receiving events has to be written in JavaScript. It is interpreted using the SpiderMonkey³ library inside the main *dSS* process. The available *API* to access the *dSS* functionality is quite limited. That's why some workarounds were needed to implement the functionality required for this thesis.

2.3 Electricity Markets

In the last decade many countries removed the strict regulations of electricity markets [6]. The price of electricity or energy in general is now determined by the economic rule of supply and demand. Let me give some insight into this topic, that is required to understand all aspects of the system. Because of the fact that electricity can not be stored efficiently, a system operator is needed that matches supply and demand [8]. Stakeholders of energy markets are:

- Electricity generators.
- Electricity providers that sell electricity to households for fixed prices (day and night tariff).
- Speculators.
- Large companies with large energy needs that directly buy their energy on the market.

¹http://developer.digitalstrom.org/download/dss/1.4/dss-1.4.2-doc/dss-1.4.2-json_api.html

²http://developer.digitalstrom.org/download/dss/1.4/dss-1.4.2-doc/dss-1.4.2-soap_api.html

³<https://developer.mozilla.org/en/SpiderMonkey>

The price is now determined by matching offers from generators, or stakeholders that want to sell electricity, to bids from consumers or stakeholders that want to buy electricity.

An electricity generator stakeholder can now place offers on the markets starting at its minimal needed price to still make profit. But of course the generator can also buy other electricity that may enables it to shut down its generators because it can buy the energy it has to deliver cheaper than to actually produce it. As you can see the system is very complex.

There are three types of energy markets I would like to give a very short description of⁴.

2.3.1 Derivatives Market

This market is intended for long-term trades. In the year 2011 you can already buy energy that will be consumed in 2015. The amounts of traded electricity on this market is very large and the prices are rather stable.

2.3.2 Spot Market

On the spot market the stakeholders trade energy for about the next week. The prices on this market are still quite stable.

2.3.3 Intraday Market

The intraday market serves electricity requests and offers for about the next 24 hours. The prices of this market underlie large fluctuations as seen in the figure 2.4. There sometimes even occur negative electricity prices because it is cheaper for an atomic power plant to pay for its produced energy than to shut down the reactor. The traded volume on the intraday market is very small compared to the derivatives market.

⁴<http://www.eex.com/en/Market%20Data>

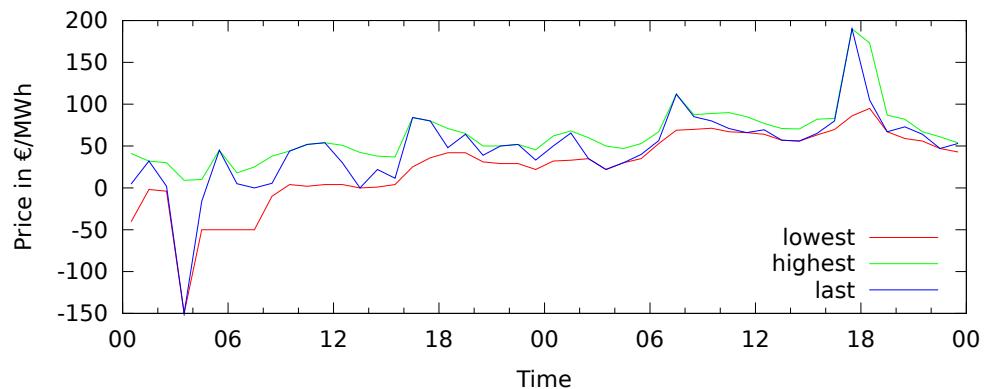


Figure 2.4: Price curves in the time between 27.11.2011 00:00 and 28.11.2011 24:00⁵ on the German electricity market. Note that such low negative prices are very rare. Normally the prices stay positive.

⁵<http://www.epexspot.com/en/market-data/intraday/intraday-table/2011-11-27/DE>

CHAPTER 3

Smart-Grid App

This chapter describes the smart-grid app I developed for the *dSS*.

3.1 Idea

The user has to have full control over everything that is sent to the energy provider. This principle should facilitate the acceptance by the users and is important for privacy issues as discussed further in the section 6.3.

The user specifies devices to which the energy provider has access. The energy provider then can remotely start up or shut down the configured devices according to the rules the user specified. The compliance with the rules is enforced by the local digitalSTROM installation. But the energy provider has all the freedom to act within the specified rules.

3.2 Configuration

The user of the digitalSTROM installation has to configure the devices in the configuration web *UI* of the smart-grid app, as seen in figure 3.1.

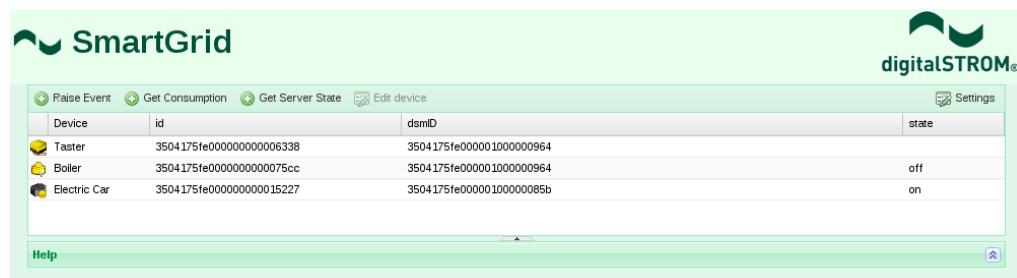


Figure 3.1: The web configuration *UI* start screen of the smart-grid app.

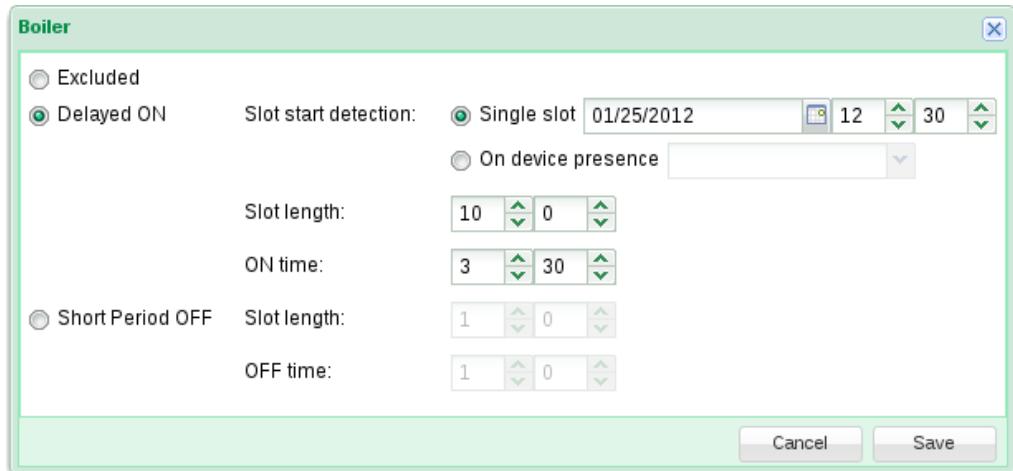


Figure 3.2: Configuration screen of a single device.

Using the configuration pop up as seen in figure 3.2 each device in the digitalSTROM installation can be configured to be in one of the following three states according to the smart-grid algorithm:

- Excluded from the algorithm. The device will always work no matter what the energy provider does. The energy provider is not going to receive information about this device.
- Delayed ON: a device in this state is normally OFF. If it is used its start may be delayed by the energy provider according to the rules specified.
- Short period OFF: a device in this state is normally ON. It can be powered OFF by the energy provider for a short time if there is not enough energy available.

We now describe the last two states in more detail:

3.2.1 Delayed ON

This group contains devices of which the start time is not that relevant. For example if we have an electric car we don't actually care when it is charged but we care that it is charged in the morning when going to work. Other typical devices that could be configured with this state are:

- Boiler
- Washing machine / Tumbler
- Dish washer

A delay ON device is given a time slot that has to be longer than the time the device needs to be powered ON. To simplify the configuration a device cannot be powered OFF if it is once started until the minimal ON time is reached. The system's task is now to ensure that the device is at least powered ON for its minimal ON time during the slot given. The parameters that have to be specified for a device in this group are listed in the following table:

Parameter	Description
Slot start	The slot start can either be given as an absolute time or dynamically by detecting the presence of a selectable clamp. The second possibility is discussed in the section 3.6.
Slot length	The length of the slot.
ON time	The minimal time for which the connected device should be powered ON during the slot.

3.2.2 Short OFF

Short OFF devices are normally powered ON but it does not matter if they are shut OFF for a short amount of time. A typical short OFF device is an electric heater that does not need to heat exactly at the consumption peek. Normally it does not matter if its powered OFF for some minutes. The room will not cool down very much during that time. Other typical devices that could be configured short OFF are:

- Heat pump
- Freezer
- Air conditioner

A short OFF device is given a slot length and a maximal OFF time in this slot. The system then ensures that the device is powered OFF for maximally the given amount of time inside the slot. The slot is automatically repeated after its expiration. The following table shows the parameters of a short OFF device:

Parameter	Description
Slot length	The length of a slot.
OFF time	The maximal time the device can be powered OFF during a slot.

3.2.3 Example Configurations

The device configured with the parameters stated in the table 3.1(b) is a device that is normally ON. In an interval of six hours it can be turned OFF for maximally twenty minutes.

(a) Car charging		(b) Freezer	
Parameter	Description	Parameter	Description
Type	Delay ON	Type	Short OFF
Slot start	Jan 25, 2012 17:30	Slot length	6 hours
Slot length	12 hours.	OFF time	20 minutes
ON time	4 hours		

Table 3.1: Example configurations of some devices.

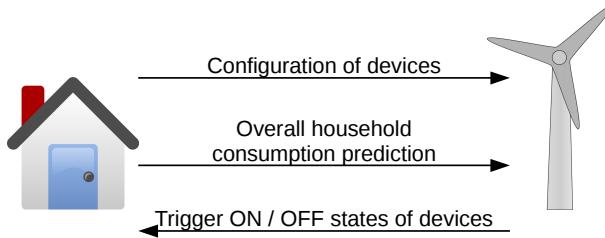


Figure 3.3: Visualization of the protocol.

The device configured with the parameters stated in the table 3.1(a) is a device that is normally OFF. The slot of the device starts on January 25, at 17:30 and ends 12 hours later on January 26, at 05:30. In this slot the device has to be powered ON for at least 4 hours. Its latest starting time is therefore January 26, at 01:30.

3.3 Dataflow

The protocol between the building and the electricity provider is sketched in the Figure 3.3. An outline of the protocol would look like this:

1. The building sends its configuration values for all devices, that are configured short OFF or delay ON, to the energy provider. If some sort of consumption pattern information of the devices is available this is sent as well.
2. The building periodically sends a prediction of the consumption for the next time unit to the energy provider.
3. The energy provider computes commands that will be sent to the building to trigger its configured devices.

This provides a mechanism for the energy provider to control the devices configured by the user. The prediction of the household consumption can be used at the provider's side to improve the overall consumption prediction.

3.3.1 Protocol

The protocol I implemented is a very simple *Extensible Markup Language (XML)* based protocol. Because of the limited JavaScript interface provided by the *dSS*, polling had to be implemented. We got it to the point that a connection to the energy provider is kept open all the time. Most household's networks these days are not directly accessible through the internet but are hidden behind a *Network Address Translation (NAT)* device. Because of this they are not directly accessible from the internet. To avoid the *NAT* problem the *dSS* app opens a connection to the energy provider and sends its device configurations. This punches a hole through the *NAT* and the energy provider can now send data to the household's *dSS* over that connection. Of course other techniques discussed in [5] could be used to avoid *NATs*. After opening the connection and sending its configuration the *dSS* app cannot send data over this connection because of the interface limitation. It just can send directly after receiving data from the energy provider over it. This problem occurs because of the JavaScript scopes. On every JavaScript execution a new scope is created. There is no way to share open connections between scopes. On receiving data the scope that was used to create the connection is called again and the connection can be used again to send data. That's why the energy provider needs to poll the *dSS*. One could also implement a busy waiting scheme in the app but I decided for the polling to save resources on the *dSS*. Additionally the polling ensures that the connection stays open. To ensure that always a connection is open the *dSS* periodically checks for the connection to be open (achieved with timestamps that are stored in the property tree discussed in section 3.4) and opens a new one if it was closed before.

First the household and the energy provider need to make sure that their clocks are more or less synchronized. Since the *dSS* gets its time via *Network Time Protocol (NTP)* this is just implemented as a check. If the check fails we just wait for some time and then retry the check. The check is just executed at the beginning of the protocol assuming the time to be correct afterward. The check is initiated by the household directly after opening the connection. The household asks the server for its time with the message: `<time/>` and the server answers with: `<time>Mon, 30 Jan 2012 09:09:29 GMT</time>`. If the time difference does not exceed ten seconds the client send the configuration of its devices. After that the server starts to poll the household with the message `<ping/>`. The household replies on a `<ping/>` with either actual consumption information or new device configurations. The message for new consumption information or prediction looks like this:

```
<consumptions>
  <item>
    <timestamp>
      Mon, 30 Jan 2012 09:33:01 GMT
```

```

        </timestamp>
        <value>31</value>
    </item>
    ...
</consumptions>
```

and the message for new device configurations like this:

```

<config>
    <item>
        <type>off</type>
        <id>3504175fe0000000000075cc</id>
        <slotLength>3600</slotLength>
        <offTime>3540</offTime>
    </item>
    <item>
        <type>on</type>
        <id>3504175fe000000000015227</id>
        <startTime>Mon, 30 Jan 2012 10:38:43 GMT</startTime>
        <slotLength>3600</slotLength>
        <onTime>60</onTime>
    </item>
    ...
</config>
```

The id transmitted in the message is the *dSID* of the configured device used to identify the device in the household.

3.4 Property Tree

The only way to conserve data between multiple runs of the JavaScript interpreter is to save the data in the so called property tree. This is an *XML* file that is accessible with some helper function of the internal JavaScript *API* or the *JSON / SOAP APIs*. I used the property tree extensively in the smart-grid app because it is the only way to store data between scopes. The property tree node of my app contains the following entries:

Entry	Description
serverAddress	Address of the server of the energy provider to connect to.
serverPort	Port of the server of the energy provider to connect to.
pollInterval	Time that specifies when the next check for a still available connection to the energy provider should happen.
startupPerformed	This entry is never written to the xml file but just kept in the cached version of the property tree by the <i>dSS</i> . It is used to check whether the script is launched for the first time.
clockDriftOK and checkingClockDrift	These entries are used to make sure that the energy provider's and the <i>dSS</i> 's clock are more or less synchronized.
timeLastDataReceived	Used to check if the connection to the energy provider is still open.
sendRequest	A flag to signal that the configuration of a device has changed. The next time the <i>dSS</i> is polled by the energy provider it should send its new configuration if this flag is set.
pollEventId	Used to store the event id of the delayed poll event if the poll interval changes and the event has to be rescheduled.

Of course also all the configuration of the devices need to be stored in the property tree. Each device has its own node inside the app node. In this device node all the information about the device is stored.

The property tree can be viewed through the web user interface of the *dSS* under **SW: System > System Properties**.

3.5 The Configuration User Interface

Like the main user interface of the *dSS* also my app builds on the JavaScript ExtJS4 framework by Sencha¹. ExtJS4 is a very rich JavaScript library² that introduces known concepts of object oriented programming, like classical inheritance and mixins, to JavaScript. It comes along with many ready to use UI components. To use the same library simplified many tasks: for example I could directly use the template of the already existing digitalSTROM apps. The web interface communicates with the *dSS* via the JSON interface. With the JSON

¹<http://www.sencha.com>

²<http://docs.sencha.com/ext-js/4-0/>

interface it is directly possible to read and write the property tree as well as raising events that may trigger executions of app JavaScript code on the *dSS*.

3.6 Detecting Device Presence

For the delay ON devices I implemented a mechanism to start a new slot automatically if a specified device is connected to a power plug. For example the system can detect that the electric car has been connected to a power plug and then automatically start the slot with the configured ON time and slot length. The idea was to have a digitalSTROM clamp between the device and the power plug that is disconnected from the power line if the device is disconnected. Then the system can check the presence of that clamp and react accordingly. The presence detection of a clamp turned out to be not trivial. Because of the limited bandwidth on the power line the presence of devices is just checked once a day. This means that the *dSS* normally does not know if a device is present or not. But the *dSS* can communicate via the *dSMs* with the clamps. So a presence detection of a clamp can look like this:

1. Ask the corresponding *dSM* of a clamp via RS485 about the current status of the clamp.
2. The *dSM* asks the clamp over the power line.
3. If the clamp is present it answers to the *dSM* with its current status.
4. If an answer is received on the *dSM* it is sent back to the *dSS* over the RS485 bus.

If the clamp answers then it has to be present. If it does not answer this does not mean that it is not present. There could have been a collision on the power line leading to packet loss from the clamp. But if it still not answers after another try then it is likely that the clamp is not present. Unfortunately the limited *API* hindered the straight implementation of this idea. There is currently no function in the internal app JavaScript *API* to get the status of a clamp, but there is one in the *JSON API*. So I had to find a way to access the *JSON API* from inside the *dSS*. From outside, the *JSON API* is accessible through an SSL encrypted HTTP channel. From inside, the *JSON API* can be contacted over an insecure HTTP channel but a token is needed to get access to all the functionality. This token can be retrieved over the insecure channel as well by authenticating with username and password. This means I had to build up HTTP requests that were sent over a TCP connection to localhost. From the replies parsed with a *JSON* parser the data could be extracted.

The way this problem was solved is not very nice but there is currently no better solution.

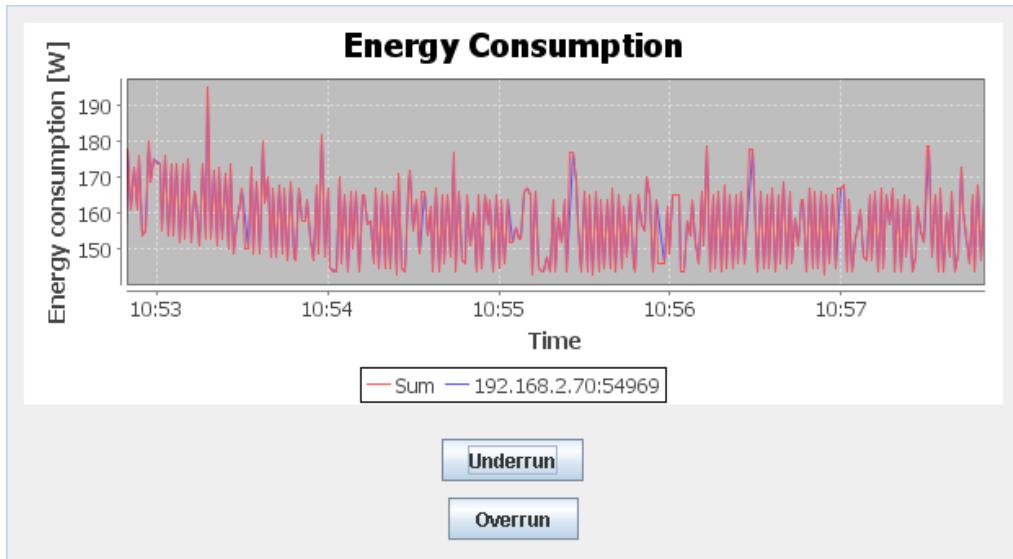


Figure 3.4: Screen shot of the energy provider demo application. Because there is just one client connected the consumption curves for the sum and the single client cannot be differentiated. The library that was used to display the graph is called JFreeChart³.

3.7 Demonstration: Energy Provider

In order to demonstrate that my app works I implemented a very simple energy provider server in Java. There is no real logic in the server but just two buttons, one for over and one for under run. A press to the under run button starts every device that can be started at the moment opposed to the overrun button that shuts everything down that is possible. Apart from that the application collects the consumption information from the connected clients. See a screen shot of the servers user interface in figure 3.4.

3.8 Deploying the System

During the work we used a digitalSTROM installation box that was provided by aizo. The box contained everything that was needed for the development of the app. A picture of the demonstration box can be seen in the figure 3.5.

To test the system in a more realistic environment we deployed it in the demonstration apartment of aizo. In the demonstration apartment we configured two devices to be used with the system. The first device was an Electrolux

³<http://www.jfree.org/jfreechart/>

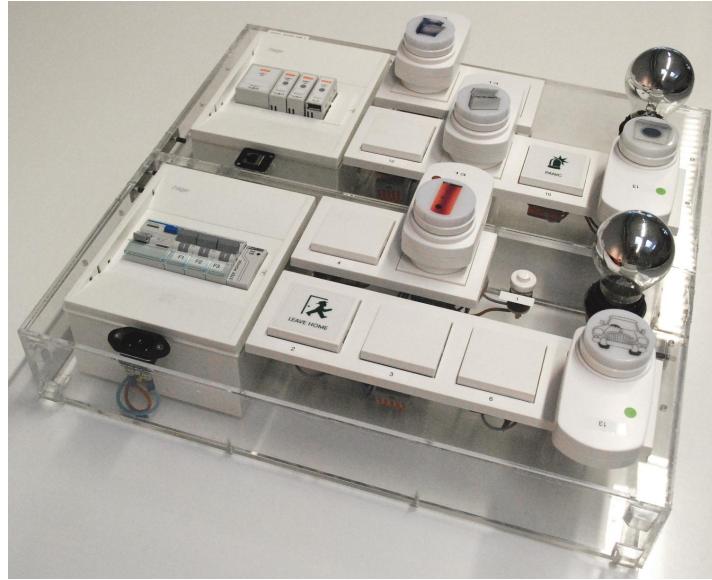


Figure 3.5: Demonstration box containing switches, light bulbs and power sockets. The top left fuse box contains a *dSF*, two *dSMs* and a *dSS*. The bottom left fuse box contains an earth leakage circuit breaker, some fuses and the power supply for the *dSS*.

GT234N freezer⁴ equipped with a computer readable temperature sensor. With this setting we could monitor the temperature in the freezer during the test period. The other device was a Segway electric roller⁵. Photographs of the test setting can be seen in figure 3.6.

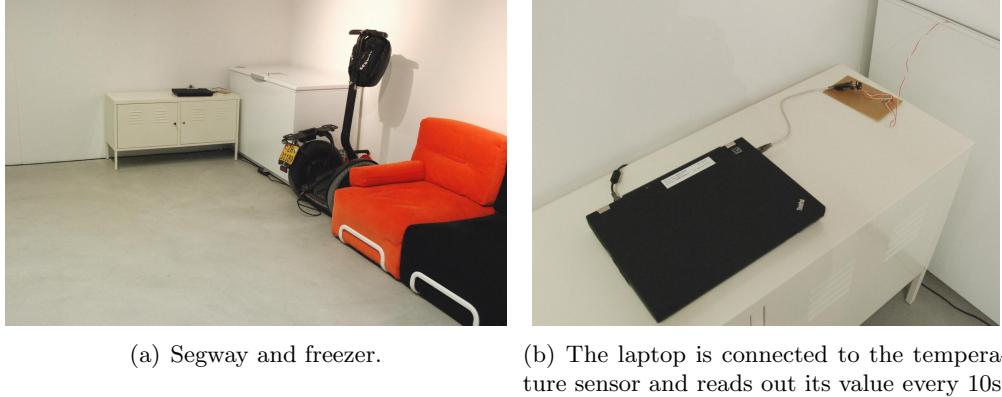
The freezer was configured with the parameters listed in the table 3.2(b) and the Segway with those of table 3.2(a).

⁴<http://www.electrolux.ch>

⁵<http://www.segway.ch>

(a) Segway		(b) Freezer	
Parameter	Description	Parameter	Value
Type	Delay ON	Type	Short OFF
Slot start	on presence detection of Segway clamp	Slot length	1 hour
Slot length	20 hours.	OFF time	10 minutes
ON time	4 hours		

Table 3.2: Configuration of the demonstration devices.



(a) Segway and freezer.

(b) The laptop is connected to the temperature sensor and reads out its value every 10s.

Figure 3.6: Deployment in the demonstration apartment.

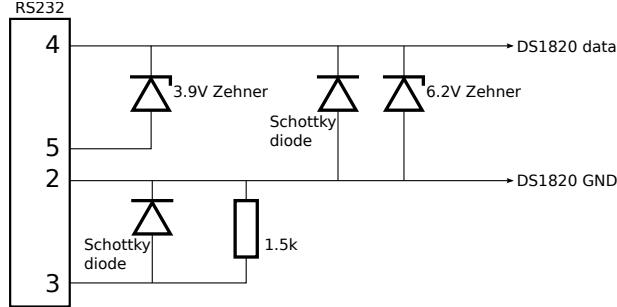


Figure 3.7: Schema to connect the DS1820 temperature sensor to the serial port.

The temperature sensor used was a DS1820 sensor⁶ connected to the serial bus of the computer using the schema in the figure 3.7. The open source software to read out the temperature value of the sensor is called digitemp⁷.

The devices worked as expected. For the Segway the only criterion the system had to fulfill was that it was charged within 20 hours after plugged in. Because the Segway was not used very much during this time the results were not very informative. But it was always charged when used.

The freezer could be analyzed much better because there was temperature information available. The freezer was configured to -20°C . First the figure 3.8(a) shows the temperature of the empty freezer without the load-balancing system working. As you can see the freezer is cooling in almost equidistant intervals keeping the temperature below -20°C . The figure 3.8(b) shows the temperature curve of the same freezer but this time it was filled with thirteen 1.5 liter bottles of water. As you can see the cooling intervals are now slightly longer because water is a much better cold accumulator than air. The figures 3.9(a)

⁶<http://datasheets.maxim-ic.com/en/ds/DS18S20.pdf>

⁷<http://www.digitemp.com/>

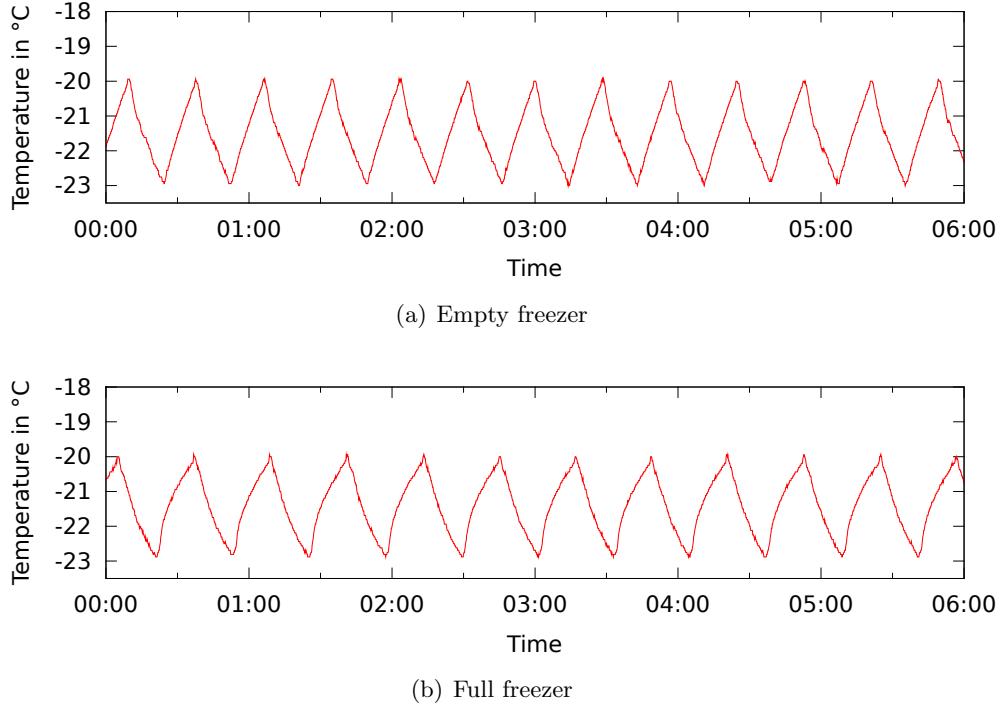


Figure 3.8: Freezer temperatures without load-balancing.

and 3.9(b) show the temperatures with activated load-balancing algorithm. The freezer was powered OFF manually, using the demonstration energy provider application, for 10 minutes at the times:

- 11:06:23
- 12:59:13
- 14:21:47
- 15:34:39

The maximal temperatures reached in the experiment were about -18.2°C with the empty freezer and -18.9°C with the full freezer. These values were reached by powering OFF the freezer exactly at the time it wanted to start cooling. Depending on the purpose of the freezer the user has to decide whether these temperatures are acceptable or not.

To actually use the load-balancing system with a freezer there should be a way to bind it to the temperature inside the freezer. The experiments did not take into account that a freezer may be opened. Opening the freezer leads to a large increase in temperature that should be balanced out as fast as possible. If

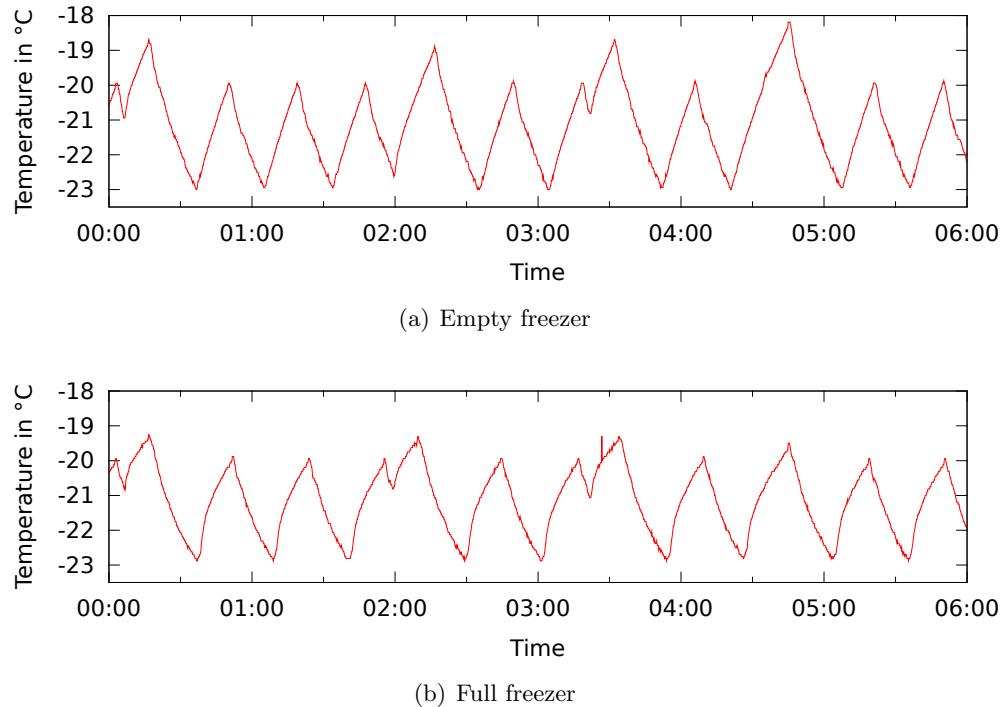


Figure 3.9: Freezer temperatures with load-balancing.

the freezer is exactly powered off in this time this could increase the temperature to a level that damages the goods inside the freezer. So the system should be deployed directly in the freezers software to have a way to take the temperature into account.

CHAPTER 4

Load-Balancing Algorithms

In this chapter we want to address the problem of actually using the device data provided by the user at the energy provider. There are many possibilities to use this information. In our two approaches we focused on the principle of using the available energy. This is also what would increase the user's acceptance. An energy provider could also try optimize the problem with the focus on other criterion, like the increase of its winnings. However in this thesis we only focused on the constraint to use the available energy as efficient as possible. Under this point of view the optimal load-balancing algorithm minimizes this equation:

$$\int_{t=0}^{\infty} |\text{availability}(t) - \text{consumption}(t)| dt \quad (4.1)$$

4.1 Simple Reacting Algorithm

The idea of this first algorithm is to just react on under / over runs as seen in figure 4.1. No prediction is involved. Just the actual availability and consumption values are compared and actions are taken based on these.

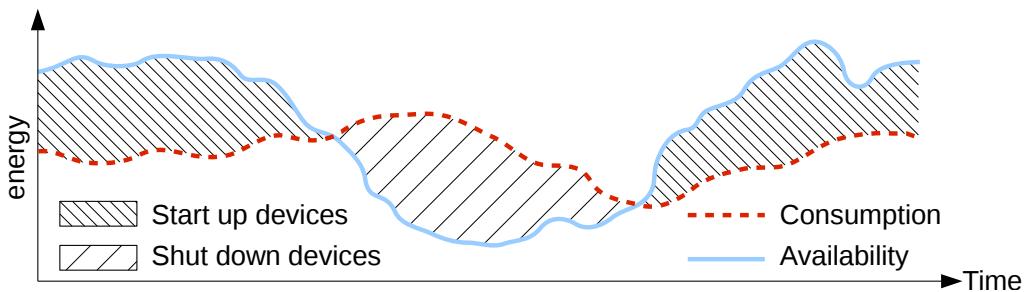


Figure 4.1: Illustration of the idea of the simple algorithm. Just the availabilities and the consumptions are compared.

```

1 overrun := false;
2 underrun := false;
3 measure consumption and availability;
4 delta := abs(consumption - availability);
5 number := delta / avg_device_consumption;
6 if consumption > availability then begin
7     for i := 0 to number do begin
8         if a_device_can_be_turned_off then begin
9             turn_off_that_device;
10        end;
11        else begin
12            overrun := true;
13            Break;
14        end;
15    end;
16 end;
17 else begin
18     for i := 0 to number do begin
19         if a_device_can_be_turned_on then begin
20             turn_on_device;
21         end;
22         else begin
23             underrun := true;
24             Break;
25         end;
26     end;
27 end;
28 if overrun then begin
29     turn_off_power_plants;
30 end;
31 else if underrun then begin
32     turn_on_power_plants;
33 end;

```

Listing 4.1: Pseudo code of a reacting algorithm.

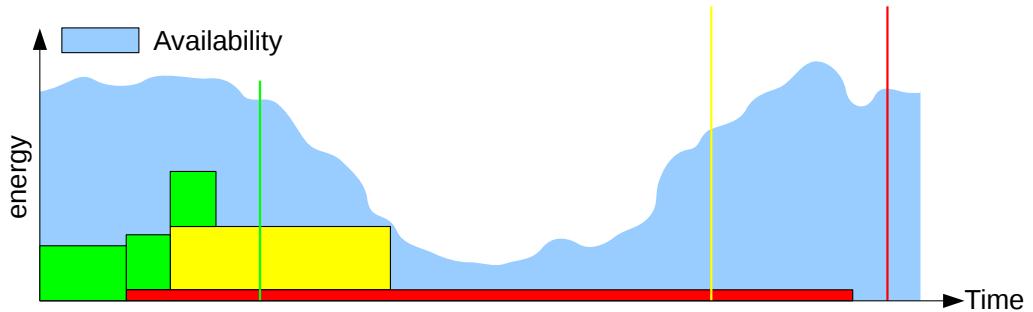


Figure 4.2: The devices have to be scheduled to be finished before their deadlines (vertical lines). The rectangles represent the energy consumption period of a device.

The algorithm just needs measurements of the availability and the consumption and an estimation of the average consumption of a device. If there is consumption information available for the specific devices one does not have to rely on the average consumption estimation but instead directly use the consumption information of the device to turn ON / OFF. The running time of the algorithm depends on the underlying data structures. With n devices the search for a device that can be turned ON / OFF takes $O(\log(n))$ if the devices are stored in an interval tree where the intervals are the times when a device can be turned ON / OFF. The worst case is if we have to find n devices to turn ON / OFF. Therefore we get a running time of $O(n \log(n))$. There may be better data structures but even with an interval tree the running time is acceptable.

4.2 2D Packing Algorithm

Another algorithm I investigated is an adapted version of the 2D bin packing problem as described in [12] and [13]. The handling of the “short OFF” devices does not differ from the simple algorithm but now the “delay ON” devices are handled differently. On receiving the data of a “delay ON” device this device is scheduled at the first possible time in its starting interval where enough energy is available as seen in figure 4.2. To do this we need a prediction of the future availabilities and consumptions. A real energy provider would use its well-proven prediction methods to get this information. In this thesis we used consumption information of the past from eex.com¹ to simulate the algorithm. The algorithm is sketched in the Listing 4.2. The hardest part is to find the best starting point for a device. Only heuristics are available here because it is an even harder 2D bin packing problem. Note that we don't have a rectangle to package into but the area bounded by the x-axis and the function:

¹http://www.transparency.eex.com/de/daten_uebertragungsnetzbetreiber/stromerzeugung

```

1 run_the_simple_algorithm_ignoring_delay_on_devices;
2
3 for i := 0 to length(new_delay_on_device) do begin
4     device := new_delay_on_device[i];
5     start_at := find_best_starting_point(device);
6     device.turn_on(start_at);
7 end;

```

Listing 4.2: Pseudo code of a scheduling algorithm.

$$f(t) = \text{prediction_of_available}(t) - \text{prediction_of_consumption}(t) \quad (4.2)$$

4.2.1 Heuristic to Find the Best Starting Point

The device has to be scheduled in the following interval:

$$I = [\max(\text{now}, \text{earliest_starting_time}), \text{latest_starting_time}]$$

A good heuristic is to start the device at the first point in the interval where $g(t) = f(t) - \text{consumption_of_device}(t)$ is positive. There are 3 possibilities where this first positive point can occur.

1st case $g(\max(\text{now}, \text{earliest_starting_time}))$ is positive. We are done.

2nd case $g(t)$ does have one or multiple zero points in the interval I . We can compute one of them assuming $g(t)$ to be continuous with the Newton method. We have to repeat the Newton method multiple times with changing intervals to make sure that we found the smallest zero point.

3rd case $g(t)$ is not positive in the interval I . In this case we apply another heuristic. We first compute a local maximum of $g(t)$ in the interval I . Again assuming the $g(t)$ to be continuous we can apply a binary search on the interval. Now we know that at that point where $g(t)$ is maximal the device should run because it would have the least impact on the system. The actual starting point

is then computed like this:

$$\begin{aligned}
 t &= \text{timepoint_in_}I\text{_with_largest_availability} \\
 e &= \max(\text{now}, \text{earliest_starting_time}) \\
 l &= \text{latest_starting_time} - e \\
 f &= \begin{cases} 1 - (t - e)/l & \text{if } t - e > l/2 \\ (t - e)/l & \text{otherwise} \end{cases} \\
 \text{start_at} &= t - f \cdot \text{onTime}
 \end{aligned}$$

Intuitively the starting point of the device is computed by taking the point with largest availability and subtracting a factor of the ON time of it because the device should already run at this time point. The factor depends on the position of the largest availability point in the interval I .

4.3 Simulating the Algorithms

We did not have the possibility to deploy the system in a real village or town, that is why we simulated the algorithms. Our simulations are not that close to reality because of the lack of information and data. Some shortcoming of the simulations are:

- Devices are simulated as static machines that consume a constant amount of energy if powered ON and nothing if powered OFF. Especially short OFF devices consume a constant amount of energy if they are not used to load balance.
- If no algorithm operates on the system there is a constant consumption. In reality the consumption varies during a day.

Nevertheless it is possible to compare different algorithms in a quantitative way with this simulator.

To simulate the 2D packing algorithm predictions of the future availability and consumption are needed. The availability of the simulation is determined by a static function. The availability function was taken from the average energy production of Germany and Austria published by eex.com² for the time between November 21, 2011 and December 12, 2011 as seen in figure 4.3. The availability data was scaled down to be able to consume all the energy in our simulations. To predict the availability the same data was used leading to an exact prediction. The prediction of the consumption was computed by determining the average consumption if no algorithm is influencing the system. To this value the consumption of already scheduled delay ON devices was added.

²http://www.transparency.eex.com/de/daten_uebertragungsnetzbetreiber/stromerzeugung/

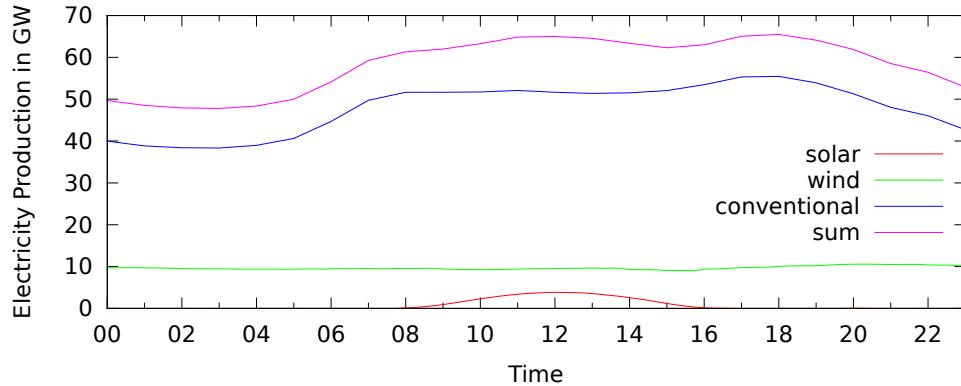


Figure 4.3: Average of the electricity production between November 21, 2011 and December 12, 2011.

We simulated the algorithms for 200, 2'000, 20'000 and 200'000 devices. These numbers represent a small village, a large village, a small and a large town. About $\frac{1}{3}$ of the devices were excluded from the algorithm, $\frac{1}{3}$ were delay ON and the last $\frac{1}{3}$ were short OFF devices. There is no particular reason why one should choose $\frac{1}{3}$ of the devices. In a real environment the percentage of configured devices may be much smaller. But we wanted to have clearly visible effects on the consumption characteristics in the simulations. The overall consumption of the whole system was evaluated every 60 seconds. The parameters of the devices were assigned randomly in the following ranges:

“delay ON” devices:

Parameter	Range
Consumption	[0W, 2000W].
ON time	[1min, 4h]
Slot length	[ON time, 8h]
Next slot start in	[0, 24h]

“short OFF” devices:

Parameter	Range
Consumption	[0W, 2000W].
OFF time	[1min, 4h]
Slot length	[OFF time, 24h]

Excluded devices: These devices have a random consumption in the interval [0W, 2000W]. They change their ON / OFF state for each evaluation point with

the probability $p = 0.9$.

4.3.1 Results

In the figure 4.4 you can see the results of the simulation for 3 days. The figure 4.5 shows the function $\Delta(t) = \text{availability}(t) - \text{consumption}(t)$ for both algorithms. Finally the figure 4.6 shows the function

$$\int_{t=0}^x |\Delta(t)| dt$$

Both algorithms can not fulfill all the peak situations. But this was expected because there is a moment when all devices are turned ON or OFF and there are no more possibilities to do further corrections on the consumptions. The first peak can be served but then for the second peak no more devices are left that could be powered on in both algorithms. As you can see the simple algorithm performed better with many devices than the more sophisticated packing algorithm. This is mainly because of the fact that with many devices the expectations match reality closer than with few devices. On the contrary, the packaging algorithm performs slightly better with fewer devices. This is because with few devices the scheduling of the delay ON devices is much more important.

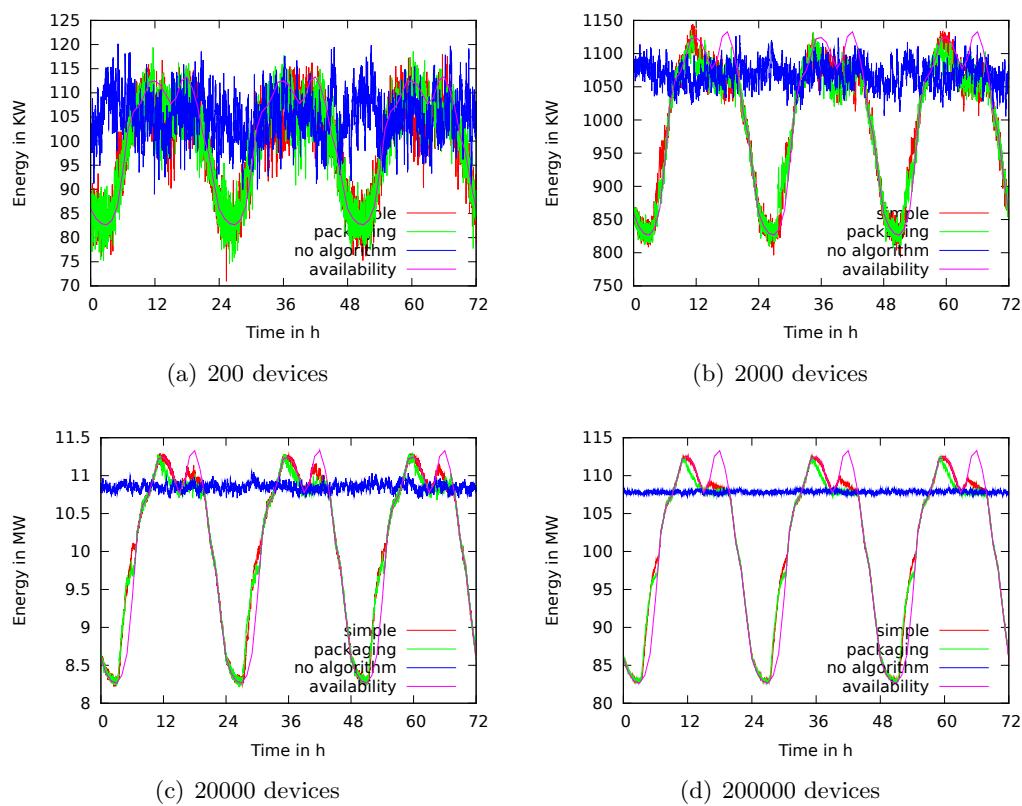


Figure 4.4: Results of simulation for four days

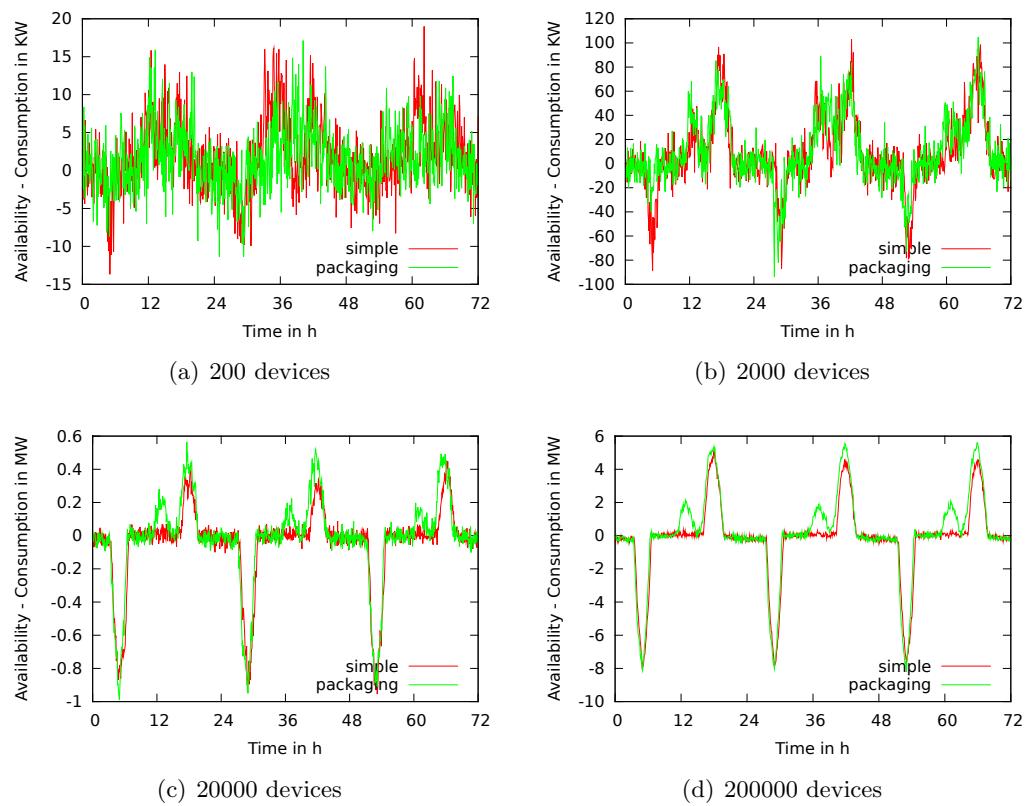


Figure 4.5: availability(t) – consumption(t) for four days

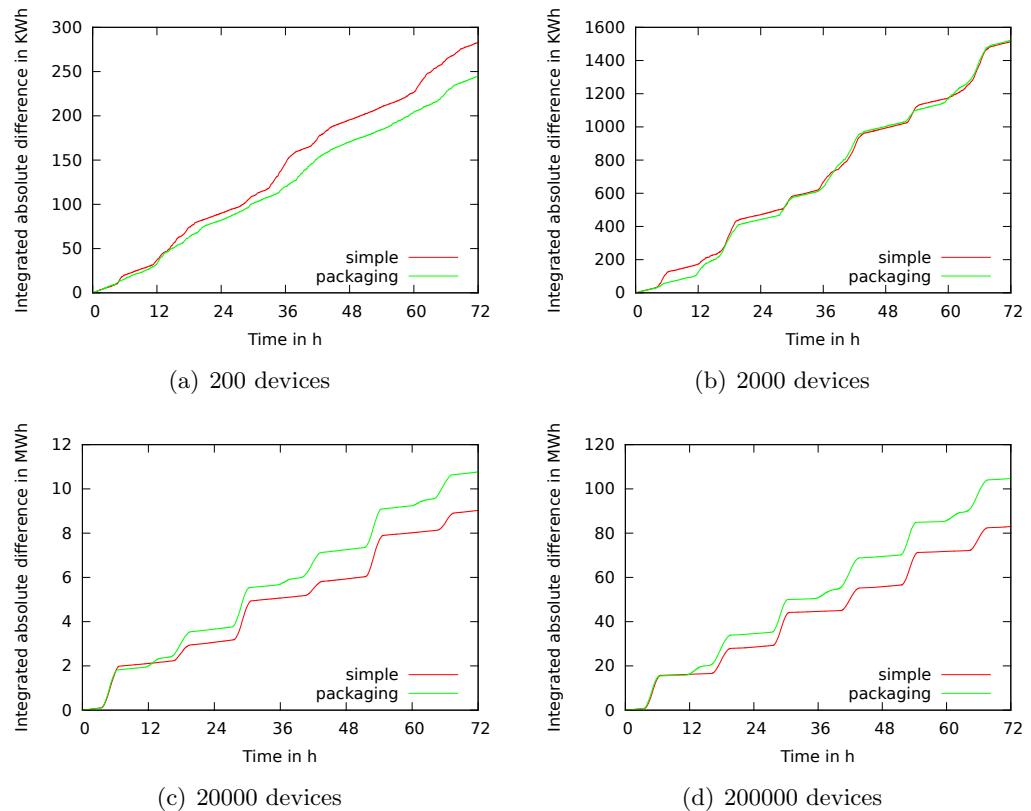


Figure 4.6: Integral of the absolute difference for four days. This represents the amount of energy that could not be compensated by the algorithms.

CHAPTER 5

Device Detection and Consumption Prediction

To be able to predict the energy consumption of a household there is a need to detect different devices with their consumption patterns. If the house knows all the details about its devices it can compute a very accurate consumption prediction and support the energy provider in predicting the overall consumption. The devices that have the most impact on the overall consumption of the house, like heater, freezer and air conditioner, often have a very regular energy consumption pattern. This is showing the collected consumption data of a household seen in figure 5.1(a). The data for this graph was collected at the household of an employee of aizo between July 13, 2011 at 3:50:00 and July 14, 2011 at 13:05:00. There are large peaks during evening and midday of the second day. The small peaks repeating about every hour may result from a freezer. If this freezer could be detected then its pattern could be taken into account to predict the consumption.

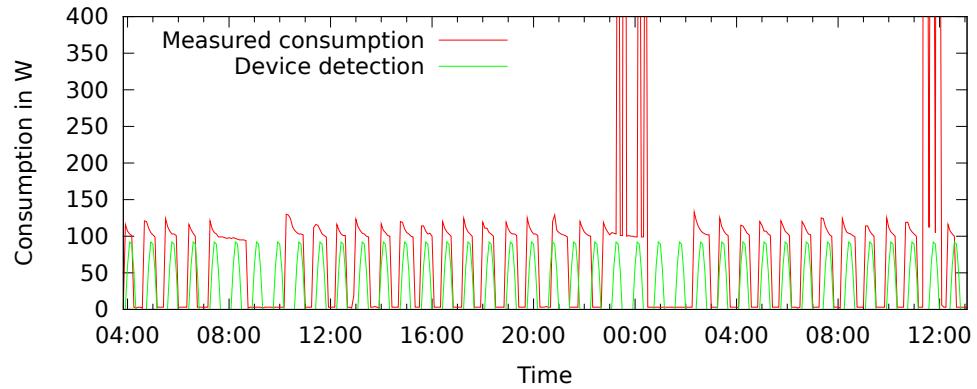
In the current version of digitalSTROM there are power meters deployed directly in the clamps. However there is currently no way to access them. This issue may be resolved in a future version of digitalSTROM. But even if there were accessible power meters at every clamp we could still have some devices that are not digitalSTROM enabled.

Additionally to the better prediction, the data could be used to detect the failure of a device. For example, the system could detect the freezer to not work anymore because of missing consumption patterns and trigger some sort of alert.

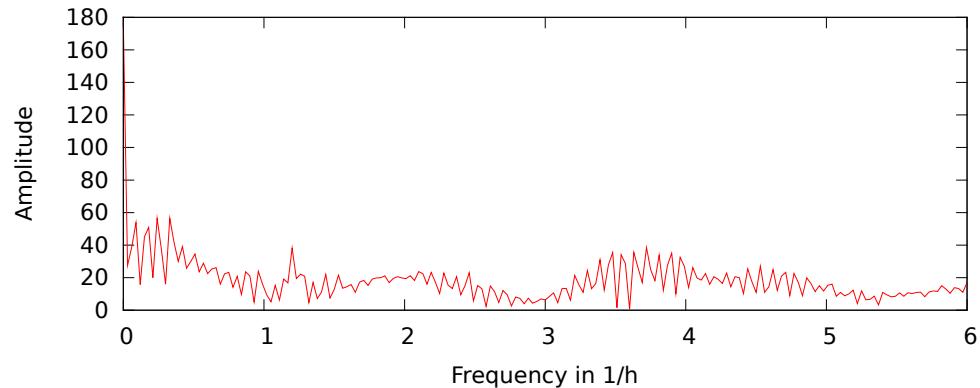
5.1 Detecting Device Consumption Pattern

The goal is to differentiate devices just by looking at the overall metering data.

One approach is to apply a discrete Fourier transformation on the consumption data. Doing this we get a vector in the frequency domain. The most



(a) Overall consumption curve and prediction curve of a single device. The prediction curve is scaled to improve visibility.



(b) Frequency spectrum.

Figure 5.1: The overall consumption curve is transformed to its frequency spectrum. From that spectrum the frequency with the most impact is extracted (about $1.2 \frac{1}{h}$) and transformed back to the time domain.

interesting frequencies are now between $0.5\frac{1}{h}$ and $2\frac{1}{h}$ because we expect device patterns in the interval $[0.5h, 2h]$. Lower frequencies originate from large peaks and larger frequencies belong to noise. Now we extract the frequency with the largest impact in the interval $[0.5h, 2h]$ and transform it back to the time domain using an inverse discrete Fourier transformation. This gives us a sine curve with the positive peaks at the moments where the device is most probably active. Multiple devices can be extracted like this and their active times extrapolated. The whole process is illustrated in the figure 5.1.

5.2 Consumption and Availability Predictions

Energy providers know these predictions very well. This is why the whole energy system works at all these days. To improve those predictions the household could compute some local predictions and send them to the energy provider. To do this as accurate as possible it would be nice to have per device consumption information about the last few days. With such information one could compare the last days per device, try to find some patterns in the consumption, and give predictions per device. The power provider is not interested in per device predictions so the delivered result would be the sum over all devices.

To find consumption information of single devices in a digitalSTROM installation the proposal of [7] could be used. An overall meter is available as well as the ON / OFF state information of the different devices (with some delay however).

A possible model to predict the consumption for the next few hours is an artificial Elman neural network. Something like this has already been done in [2]. As input we could use

- Average consumption of the last few time units.
- Local weather information.
- Personal calendar information.
- Holiday information.
- digitalSTROM events like “going”.

CHAPTER 6

Social Aspects

6.1 Sales Appeal

In this section we try to answer the question what appeals could be created to encourage households to participate the system. The only one actually profiting from such a system directly is the energy provider. Of course a single household also wins from the system because fewer power outages may happen. But this is just a weak indirect benefit because the system cannot guarantee that no blackouts occur. Therefore the energy provider has to share its benefit with its customers. Otherwise very few would see a reason to participate.

How to share the benefits is now a question for sales people but we also made some thoughts about it. A possibility would be to grant price reductions either proportional to the number of devices that are configured to load-balance or proportional to their power consumptions. For the latter exact information has to be available for the devices. With such a stimulation the households that have the system installed are animated to configure and use as many devices as possible with the system.

Another appeal to participate in such a project would be to increase the awareness of sustainability. Who does not want its own children or grand children to live in the same world as we do these days. To ensure this we have to fundamentally change our habits and consumer behavior. A load-balancing system can not do this but it can help to go forward into the right direction.

6.2 Security

At the moment the implementation of the system does not use industry standard security mechanisms. All the communication between the household and the energy provider is not encrypted and there are no authentication checks performed. An attacker can read all the information sent in plain text. However he cannot do much harm to the system because all the rules are enforced on the *dSS*. Of

course an attacker can prevent the usage of household devices for load-balancing or even use them to worsen the problem.

All those security issues have to be fixed before professional use of the system. At least all the communication has to be encrypted and the energy provider has to authenticate itself to the *dSS*.

6.3 Privacy

There may be concerns about the energy provider knowing about the devices of a household. Of course the energy provider knows more about a household using the smart-grid app than about one not using it. But the household also benefits from a bonus. This bonus comes with the cost of giving information to the energy provider. With the configuration possibility of the user the information given to the energy provider can be specified very fine grained. The user can specify exactly what the energy provider should see.

The main problem about privacy is that the energy provider is able to map devices to households. This problem could be solved by using a peer to peer network among all households participating in the load-balancing system. The configuration would then be sent along a path in the peer to peer network. Every node in the network just knows where it received the packet from and where it has to send it to. A system like this is described in [10]. The crowds system works without encryption. This simplifies the deployment because no key distribution is required. The principle of the crowds system is the following: On receiving a request from another node in the crowd the node flips a coin whether to send the request to its destination or to send it to another node in the crowd. The packet may be forwarded many times until it reaches its destination. The path a packet takes to the destination is therefore random and the energy provider has no information about the origin of a packet. A configuration packet for a device could belong to any household that participates in the crowd. Every node has to record where the packets it forwards were received from, to be able to return the resulting answer on the same path. To introduce encryption such that not everyone in the crowd that receives a message can read it, the original sender can encrypt its message with the energy providers public key. This way it is ensured that only the energy provider can read the message. The other direction is more complicated because we can not tell the energy provider the public key of the sender. If it knew the public key of the sender it could create a mapping from device to household again. A possibility would be that the household includes a random key in its message. The energy provider can decrypt the key in the message and use it to symmetrically encrypt the messages it wants to send to the household belonging to the request. This ensures that the answer message can just be seen by the original sender because he is the only one that knows the key. An illustration of the protocol is given in the figure 6.1.

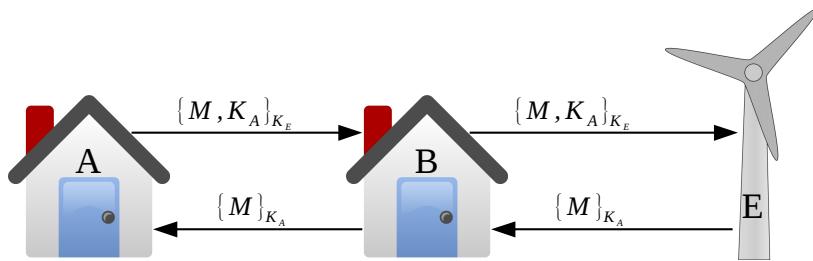


Figure 6.1: Household A generates a random key K_A and adds this key to its message. It decides to send the request to a random node in the crowd. Node B receives the message and decides randomly to send the packet to its destination. It records the route to be able to deliver the response on the same path. The energy provider E can decrypt the packet with its secret key and use the key K_A to encrypt information to A with a symmetric encryption scheme. Note that A , B , and E do not stand for Alice, Bob and Eve in this example but for A , B and E .

CHAPTER 7

Conclusions & Future Work

As the energy production has to shift towards renewable energy, we have to start thinking about load-balancing power consumption. My approach seems to be a natural extension of the ripple control system¹ [9]. It is much more flexible and supports bidirectional communication.

The experiments with the freezer and the Segway showed that the system can be used with typical household devices. Because the user can configure the devices by itself the system is very flexible and can be adapted to a very wide range of environments. There is some work needed to better integrate with the devices but in general the approach works. One serious problem is that you don't always want to set the time a device should be finished with a web *UI* from your computer. Instead the system should be integrated directly into the *UI* of devices that can be used to load-balance. For example you want to directly set the time the washing machine should be finished on the washing machine itself. To use the system with a washing machine that does not include the load-balancing system but can be powered ON / OFF with digitalSTROM the following steps are needed:

1. Start the washing machine with the desired program.
2. Power OFF the machine by switching the energy off using digitalSTROM. From the point of view of the washing machine this looks like a power outage.
3. Configure the machine to be finished at the desired time using the configuration *UI* of the smart-grid app.
4. Hope that the machine resumes the last program before the power outage when powered ON again.

This solution would not be applicable. To improve this issue an open standard, to load-balance devices, has to be defined that can interface with different bus

¹<http://www.rundsteuerung.de>

systems like digitalSTROM. Device manufacturers should be encouraged to integrate the standard into their devices. Further the system has to be tested in a real environment like the project described in [3]. The acceptance of the users and the benefit of such a system has to be evaluated.

The two algorithms developed for the energy providers side are reducing the availability consumption gap. Surprisingly the more sophisticated packaging algorithm performed worse than the simple reacting algorithm with many devices. As expected it was better with few devices. There may exist even better heuristic algorithms than the ones provided. The already available knowledge about consumption prediction at energy providers has to be used in a clever way to integrate with the new possibility to use devices in households to load-balance.

The proposed solution to detect regular device consumption patterns in overall consumption data worked well in our example data-sets for a single device. The method has to be tested with multiple devices and in larger data-sets. Further the optimal length of a data-set has to be evaluated. Unfortunately we did not have access to very much data to test this method extensively.

Bibliography

- [1] aizo ag, <http://www.digitalstrom.org/support/bedienungsanleitungen/>. *digitalSTROM Handbuch für Anwender*, 11 2011.
- [2] M. Beccali, M. Cellura, V. Lo Brano, and A. Marvuglia. Short-term prediction of household electricity consumption: Assessing weather sensitivity in a mediterranean area. *Renewable and Sustainable Energy Reviews*, 12(8):2040–2065, 2008.
- [3] D. Berner. Halbzeit beim projekt ismart in ittigen. *Bulletin des SEV VSE Including Jahresheft*, 102(9):18, 2011.
- [4] G. Dickmann. Digitalstrom®: A centralized plc topology for home automation and energy management. In *Power Line Communications and Its Applications (ISPLC), 2011 IEEE International Symposium on*, pages 352–357. IEEE, 2011.
- [5] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-peer communication across network address translators. In *USENIX Annual Technical Conference*, volume 2005, 2005.
- [6] H. Geman and A. Roncoroni. Understanding the fine structure of electricity prices. *Journal of Business, Vol. 79, No. 3, 2006*, 2006.
- [7] D. Jung and A. Savvides. Estimating building consumption breakdowns using on/off state sensing and incremental sub-meter deployment. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, SenSys '10*, pages 225–238, New York, NY, USA, 2010. ACM.
- [8] J.J. Lucia and E.S. Schwartz. Electricity prices and power derivatives: Evidence from the nordic power exchange. *Review of Derivatives Research*, 5:5–50, 2002. 10.1023/A:1013846631785.
- [9] E.R. Paessler. *Rundsteuertechnik*. Publicis MCD, 1994.
- [10] M.K. Reiter and A.D. Rubin. Crowds: anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1:66–92, November 1998.
- [11] R. Staub. digitalstrom: Gebaudeautomation mit hochvolttechnologie. *Detail*, (2):49, 2009.

- [12] L. Wei, A. Lim, and W. Zhu. A skyline-based heuristic for the 2d rectangular strip packing problem. In Kishan G. Mehrotra, Chilukuri K. Mohan, Jae C. Oh, Pramod K. Varshney, and Moonis Ali, editors, *IEA/AIE (2)*, volume 6704 of *Lecture Notes in Computer Science*, pages 286–295. Springer, 2011. <http://www.computational-logistics.org/orlib/topic/2Dx.html>.
- [13] L. Wei, D. Zhang, and Q. Chen. A least wasted first heuristic algorithm for the rectangular packing problem. *Comput. Oper. Res.*, 36:1608–1614, May 2009.

APPENDIX A

Implementation Notes

A.1 Smart-Grid App

A.1.1 Subscriptions to Events

The smart-grid app subscribes to two events. First it is called on the “running” event. This event is emitted after starting the main process of the *dSS*. Further it subscribes to the “smart-grid” event. This event is raised by the smart-grid app itself. If the event “smart-grid” is raised, the parameter `action_type` has to be set in the event. This parameter specifies the type of event that occurred. The following values of `action_type` are used so far:

Value	Description
config	The user changed the energy provider configuration of the app in the <i>UI</i> . Parameters that belong in this group are the address and port of the energy providers server.
poll	This event is raised by the script itself. On this event the script performs a check if the connection is still open.
configDevice	The user changed the configuration of a device in the <i>UI</i> .
shortOffReset	This event is raised by the script itself. On this event a short OFF device that is shut down is reactivated.
delayOnStart	The delay ON devices can be given a starting time. On their starting time this event is raised by the app itself and the device is started.
delayOnReset	This event is raised by the script if a delay ON device needs to be powered OFF again.

A.1.2 Script

The script part of the smart-grid app consists of three files:

- jsonparser.js

- rexml.js
- smartgrid.js

The first two are libraries used to parse and create *JSON* and *XML* strings. The script we implemented is contained in the file `smartgrid.js`.

A.1.3 User Interface

The *UI* part of the app consists of the following directories and files:

Folder / File	Description
dss	This folder contains the app framework provided by aizo. It uses the ExtJS4 library to create special <i>UI</i> components. Further the framework defines a unique look and feel for all apps.
ext	This folder contains the ExtJS4 library.
jsgettext	This folder contains a JavaScript implementation of gettext. With this library the app could be translated with little effort at a later time.
time.js	This is a <i>UI</i> element to display and edit a time in hours and minutes.
deviceWindow.js	This file describes the main window of the smart-grid configuration <i>UI</i> .
configWindow.js	Contains the configuration pop up window of a device.
main.js	This file is also part of the app framework by aizo. This file contains the entry method for the whole <i>UI</i> .

A.2 Demonstration Energy Provider

The simple demonstration energy provider program is written in Java. For the communication with the different clients it uses the `java.nio` library. This library can be used to do non-blocking IO operations in Java. In the following table all classes implemented are listed with their purpose:

Class	Description
Client.java	Represents a client.
Controller.java	This class implements the algorithm that controls the devices. In my demo application this algorithm is very easy.
Device.java	Contains features that are available for all devices.
DelayOnDevice.java	Extends the Device class and adds all the features needed to manage delay ON devices.
ShortOffDevice.java	Extends the Device class and adds functionality for short OFF devices.
DSSServer.java	This class manages all the communication with the clients.
EnergyProvider.java	This class contains the main method of the whole program and implements the <i>UI</i> .
Logger.java	Contains code to do nice logging.
SingletonUtil.java	Implements the singleton pattern and contains tool objects that can be reused globally.
IntradayMarketPoint.java	This class is not used any more. Its original purpose was to represent a data point from the intraday electricity market. Unfortunately the platform eex.com where the intraday market was fetched from changed its website such that the retrieval of the information would have to be reimplemented. I never used the information in my Controller class but just used it to display the actual electricity price in a chart. Because of this I decided to not reimplement the feature.

A.3 Simulation

The simulations created in this thesis were all implemented using python. In the beginning I implemented all the simulations in a single threaded design. For the 200'000 devices this turned out to be quite slow. The choice of python turned out to be a bad decision because the *Global Interpreter Lock (GIL)* made a simple expansion to a multithreaded solution impossible. The *GIL* is a mechanism in python that just allows one running instance of the interpreter per process at any point in time. This simplifies the internal implementation of python but leads to a maximal core utilization of 1 with a single process. There exists a python interpreter called Jython¹ implemented in Java that allows full multithreading. However this implementation was not really faster than the

¹<http://www.jython.org/>

single threaded version because of the huge overhead of the implementation. The only solution to work around this is to use multiple processes and *Inter-process communication (IPC)* between the processes. There are many python libraries to simplify *IPC* between processes but it is still not comparable to using threads. I implemented a solution using multiple processes and *IPC* for the simple algorithm. For the packaging algorithm this turned out to be much harder. Because of this I went back to the single core version and let the simulation runs over night. The files implemented for the simulations are listed in the following table:

File	Description
device.py	Contains classes for the three different types of devices.
simulation.py	Single process implementation of all the algorithms that were used in this thesis.
integrate.py	This simple script was used to integrate the discrete data retrieved by the simulations.

APPENDIX B

Source Code

This chapter contains all the source code that was written for this thesis. Libraries that were used are not included. See a list of listings below:

B.1 config/smart-grid.xml	6
B.2 scripts/smartgrid.js	6
B.3 ui/js/configWindow.js	21
B.4 ui/js/deviceWindow.js	23
B.5 Client.java	29
B.6 Controller.java	31
B.7 Device.java	34
B.8 DelayOnDevice.java	35
B.9 ShortOffDevice.java	36
B.10 DSSServer.java	36
B.11 EnergyProvider.java	40
B.12 Logger.java	44
B.13 SingletonUtil.java	44
B.14 IntradayMarketDataPoint.java	45
B.15 device.py	45
B.16 simulation.py	47
B.17 integrate.py	50

B.1 Smart-Grid App

B.1.1 Subscriptions to Events

```

1  <?xml version="1.0"?>
2  <subscriptions version="1">
3      <subscription event-name="smart-grid" handler-name="javascript">
4          <parameter>
5              <parameter name="filename1"/>/usr/share/dss/add-ons/smart-grid/jsonparser.js</
6                  parameter>
7              <parameter name="filename2"/>/usr/share/dss/add-ons/smart-grid/smartgrid.js</
8                  parameter>
9              <parameter name="filename3"/>/usr/share/dss/add-ons/smart-grid/rexml.js</
10                 parameter>
11             <parameter name="script_id">smart-grid</parameter>
12         </parameter>
13     </subscription>
14     <subscription event-name="running" handler-name="javascript">
15         <parameter>
16             <parameter name="filename1"/>/usr/share/dss/add-ons/smart-grid/jsonparser.js</
17                 parameter>
18             <parameter name="filename2"/>/usr/share/dss/add-ons/smart-grid/smartgrid.js</
19                 parameter>
20             <parameter name="filename3"/>/usr/share/dss/add-ons/smart-grid/rexml.js</
21                 parameter>
22         </parameter>
23     </subscription>
24 </subscriptions>

```

Listing B.1: config/smart-grid.xml

B.1.2 Script

```

/*
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 *
 * Copyright (c) 2011 digitalSTROM.org, Zurich, Switzerland
 * Author: Christof Baumann <baumachr@student.ethz.ch>
 * Based on work of: Andreas Brauchli <andreas.brauchli@aizo.com>
 */
var LOGFILE_NAME = 'smart-grid.log';
var LOG = new Logger(LOGFILE_NAME);
var LOG_PRIORITY = 10;
function log(text, priority) {
    if (priority < LOG_PRIORITY) {
        var callstack = "", caller, callerName, line;
        try {
            i.dont.exist+=0; //doesn't exist - that's the point
        } catch (e) {
            if (e.stack) { //Firefox
                //LOG.logln(e.stack);
                callstack = e.stack.split('\n');
                //Remove call to printStackTrace()
                callstack.shift();
            }
        }
        caller = callstack[0].split('(');
        callerName = caller[0];
        if (callerName === '') {
            callerName = 'callback';
        }
        LOG.logln(callerName + ': ' + text);
    }
}

```

```

42     }
43     line = parseInt(callstack[0].split('.js:')[1], 10) + 1;
44     LOG.login('[' + callerName + ':' + line + ']' + text);
45     /* var i;
46     for(i=0; i<callstack.length; i++){
47       LOG.login(callstack[i]);
48     }*/
49   }
50 }
51 */
52 /**
53 * Checks if the system version matches at least a given version
54 * @param version Version to check against (in the format X.Y.Z)
55 * @return true if the system version is at least the passed version,
56 * false otherwise, including the case where the version node doesn't exist
57 */
58 function requireSystemVersion(version) {
59   var ndSysVersion = Property.getNode('/system/version/version');
60   if (ndSysVersion) {
61     var sysVersion, reqVersion, len, i;
62     sysVersion = ndSysVersion.getValue().split('.');
63     reqVersion = version.split('.');
64     len = sysVersion.length < reqVersion.length ? sysVersion.length : reqVersion.length;
65     for (i = 0; i < len; ++i) {
66       var sys, req;
67       sys = parseInt(sysVersion[i], 10);
68       req = parseInt(reqVersion[i], 10);
69       if (req < sys || (i === len - 1 && req === sys)) {
70         return true;
71       }
72       if (req > sys) {
73         return false;
74       }
75     }
76   }
77   return false;
78 } // requireSystemVersion
79
80 function timeDriftCheck() {
81   var alreadyChecking = Property.getProperty('checkingClockDrift');
82   if (alreadyChecking !== null && alreadyChecking) {
83     log('Another instance is already checking the clock drift periodically', 3);
84     return;
85   }
86   Property.setProperty('checkingClockDrift', true);
87   timeDriftCheckImpl();
88 }
89
90 function timeDriftCheckImpl() {
91   var host = Property.getProperty('serverAddress');
92   var port = Property.getProperty('serverPort');
93   if (host === null || port === null) {
94     log('Server not specified.', 3);
95     return;
96   }
97   log('Connecting to ' + host + ':' + port, 1);
98   var socket = new TcpSocket();
99   socket.connect(host, port, function(state) {
100     if (state) {
101       log('Connected', 1);
102       socket.send('<time/>', function(bytesSent) {
103         log('Sent', 1);
104         if (bytesSent > 0) {
105           socket.receiveLine(1024, function(data) {
106             log('Received data', 1);
107             log(data, 2);
108             var xmlDoc = new REXML(data);
109             if (xmlDoc !== null && xmlDoc.rootElement !== null && xmlDoc.rootElement.type === 'element') {
110               if (xmlDoc.rootElement.name === 'time') {
111                 var now = new Date();
112                 var time = xmlDoc.rootElement.text;
113                 log('Local Time: ' + now.toUTCString(), 0);
114                 time = new Date(time);
115                 log('Server Time: ' + time.toUTCString(), 0);
116                 var drift = Math.abs(now.getTime() - time.getTime());
117                 log('Drift: ' + drift + 'ms', 0);
118                 if (drift < 10000) {
119                   Property.setProperty('clockDriftOK', true);
120                   startup();
121                 }
122               } else {
123                 log('Clock drift is too large. Waiting 60s.', 0);
124               }
125             }
126           }
127         }
128       });
129     }
130   });
131 }
132
133 
```

```

124             setTimeout(timeDriftCheckImpl, 60000);
125         }
126     }
127     else {
128         log('Root element not time.Retrying in 60s.', 0);
129         setTimeout(timeDriftCheckImpl, 60000);
130     }
131 }
132 else {
133     log('Wrong data received.Retrying in 60s.', 0);
134     setTimeout(timeDriftCheckImpl, 60000);
135 }
136 }, '\n');
137 }
138 else {
139     log('Could not send to the server.Retrying in 60s.', 0);
140     setTimeout(timeDriftCheckImpl, 60000);
141 }
142 });
143 else {
144     log('Connection to server failed.Retrying in 60s.', 0);
145     setTimeout(timeDriftCheckImpl, 60000);
146 }
147 );
148 );
149 }
150
151 function getJSONToken() {
152     jsonCall('system/login?user=dssadmin&password=dssadmin', function(data) {
153         if(data.ok) {
154             Property.setProperty('token', data.result.token);
155             log('Received JSON token', 5);
156             log('Token:' + data.result.token, 15);
157         }
158         else {
159             log('No token received', 0);
160         }
161     });
162 }
163
164 function startup() {
165     //check if we already did a proper startup (this function is called again on
166     //updateConfig)
167     var startupDone = Property.getProperty('startupPerformed');
168     if (startupDone === null || !startupDone) {
169         var ndVersion = Property.getNode('version');
170         if (ndVersion) {
171             var version = ndVersion.getValue().split('.');
172             if (version.length > 1 && version[0] === '0' && parseInt(version[1], 10) < 9) {
173                 convertOldEvents();
174             }
175         }
176         var now = new Date();
177
178         //set version number
179         Property.setProperty('version', '0.0.1');
180         Property.setFlag('version', 'ARCHIVE', true);
181         Property.store();
182
183         //check short off devices
184         getDevices().perform(function(device) {
185             var dsid = device.dsid;
186             var type = Property.getProperty('devices/' + dsid + '/config/type');
187             log(dsid + ' ' + type, 12);
188             if (type !== null && type === 'off') {
189                 var shortOff = Property.getProperty('devices/' + dsid + '/shortOff');
190                 if (shortOff !== null && shortOff) {
191                     log('is short off', 25);
192                     var time = Property.getProperty('devices/' + dsid + '/actualShortOffTime');
193                     var start = Property.getProperty('devices/' + dsid + '/actualShortOffStart');
194                     if (time === null || start === null) {
195                         log('Strange: shortOff is true but the values are not set', 0);
196                     }
197                     else {
198                         start = new Date(start);
199                         var dt = Math.floor((start.getTime() - now.getTime()) / 1000) + time;
200                         log('Starting off device ' + dsid + ' in ' + dt + 's', 3);
201                         if (dt < 10) {
202                             var shortOffSlotTime = Property.getProperty('devices/' + dsid + '/
203                                         shortOffSlotTime');
204                             Property.setProperty('devices/' + dsid + '/shortOffSlotTime',
205                                         shortOffSlotTime - dt + 10);
206                             Property.setFlag('devices/' + dsid + '/shortOffSlotTime', 'ARCHIVE',
207                                         true);
208                         }
209                     }
210                 }
211             }
212         });
213     }
214 }

```

```

204         true);
205         Property.store();
206         dt = 10; //the system has to settle first
207     }
208     var event = new TimedEvent('smart-grid', '+' + dt, {action-type: ,
209         shortOffReset, dsid: dsid});
210     var shortOffEventId = event.raise();
211
212     //not needed on crash
213     Property.setProperty('devices/' + dsid + '/shortOffEventId',
214         shortOffEventId);
215   }
216 } else if (type !== null && type === 'on') {
217   var on = Property.getProperty('devices/' + dsid + '/delayOn');
218   if (on !== null && on) {
219     //schedule delayOnReset event
220     var onTime = Property.getProperty('devices/' + dsid + '/delayOnTime');
221     var minOnTime = Property.getProperty('devices/' + dsid + '/config/onTime');
222     onTime = (new Date(onTime)).getTime();
223     var offTime = onTime + minOnTime * 1000;
224     var deltaT = Math.floor((offTime - now.getTime()) / 1000);
225     if (deltaT < 10) {
226       deltaT = 10; //the system has to settle first
227     }
228     log('Stopping on device' + dsid + 'in' + deltaT + 's', 3);
229     var stoppEvent = new TimedEvent('smart-grid', '+' + deltaT, {action-type: ,
230         delayOnReset, dsid: dsid});
231     var delayOnResetEventId = stoppEvent.raise();
232
233     Property.setProperty('devices/' + dsid + '/delayOnResetEventId',
234         delayOnResetEventId);
235   }
236   else {
237     //schedule delayOnStart event
238     var startTime = Property.getProperty('devices/' + dsid + '/config/startTime',
239         );
240     if (startTime !== null) {
241       //schedule delayOnStart event
242       var plannedOn = Property.getProperty('devices/' + dsid + '/
243         delayOnPlannedTime');
244       plannedOn = new Date(plannedOn);
245       var ddt = Math.floor((plannedOn.getTime() - now.getTime()) / 1000);
246       if (ddt < 10) {
247         ddt = 10; //the system has to settle first
248       }
249       log('Starting on device' + dsid + 'in' + ddt + 's', 3);
250       var startEvent = new TimedEvent('smart-grid', '+' + ddt, {action-type: ,
251         delayOnStart, dsid: dsid});
252       var delayOnEventId = startEvent.raise();
253
254       //not needed on crash
255       Property.setProperty('devices/' + dsid + '/delayOnEventId', delayOnEventId
256         );
257     }
258   }
259   persistentConnection();
260
261   //start polling
262   raiseNextPollEvent();
263 } // startup
264
265 function convertOldEvents() {
266   /*TODO*/
267 }
268
269
270
271 function jsonCall(path, callback, callbackArgument) {
272   var token = '';
273   var TOKEN = Property.getProperty('token');
274   if (TOKEN !== null) {
275     if (path.indexOf('?') === -1) {
276       token = '?token=' + TOKEN;
277     }
278   } else {

```

```

279         token = '&token=' + TOKEN;
280     }
281 }
282 else {
283     log('token==null', 10);
284 }
285 var data = 'GET/u/json/' + path + token + 'uHTTP/1.0\r\n\r\n';
286 log(data, 10);
287 httpRequest('localhost', 8088, data, function(line) {
288     log(line, 10);
289     var data = JSON.parse(line);
290     callback(data, callbackArgument);
291 });
292 }
293
294 function httpRequest(host, port, data, success, failure) {
295     if (failure === null) {
296         failure = function() {};
297     }
298     var socket = new TcpSocket();
299     socket.connect(host, port, function(state) {
300         if (state) {
301             log('Connected', 10);
302             socket.send(data, function(bytesSent) {
303                 if (bytesSent > 0) {
304                     socket.receive(1024, function(data) {
305                         var lineArray = data.split('\r\n');
306                         var header = true;
307                         var i;
308                         for (i=0; i<lineArray.length; i++) {
309                             if (!header) {
310                                 log('Data:' + lineArray[i], 10);
311                                 success(lineArray[i]);
312                             }
313                             else {
314                                 log('Header:' + lineArray[i], 10);
315                             }
316                             header = (lineArray[i] !== '');
317                         }
318                     });
319                 }
320                 else {
321                     failure();
322                 }
323             });
324         }
325         else {
326             log('Connection failed', 0);
327             failure();
328         }
329     });
330 }
331
332 function zeroPad(number) {
333     return (number<10) ? '0'+number : number;
334 }
335
336 function poll() {
337     var observe = Property.getNode('devicesToObserve');
338     if (observe !== null) {
339         var i;
340         observe.observe.getChildren();
341         for (i = 0; i < observe.length; i++) {
342             var dsid = observe[i].name;
343
344             var checkIt = false;
345             var targets = Property.getNode('devicesToObserve/' + dsid).getChildren();
346             var j;
347             for (j = 0; j < targets.length; j++) {
348                 var startTime = Property.getProperty('devices/' + targets[j].name + '/config/
349                     startTime');
350                 if (startTime === null) {
351                     checkIt = true;
352                 }
353
354                 if (checkIt) {
355                     log('Checking device' + dsid, 9);
356                     jsonCall('device/getConfig?class=64&index=0&dsid=' + dsid, checkDeviceCallback
357                         , dsid);
358                 }
359             }
360         }
361     }
362 }
363
364 function checkDeviceCallback(data, dsid) {

```

```

361     if (data.ok) {
362       log('Device' + dsid + 'isPresent', 5);
363       var targets = Property.getNode('devicesToObserve/' + dsid).getChildren();
364       var now = new Date();
365       var j;
366       for (j = 0; j < targets.length; j++) {
367         var target = targets[j].name;
368         var startTime = Property.getProperty('devices/' + target + '/config/startTime');
369         if (startTime === null) {
370           Property.setProperty('devices/' + target + '/config/startTime', now.toUTCString());
371           Property.setFlag('devices/' + target + '/config/startTime', 'ARCHIVE', true);
372         }
373         Property.setProperty('devices/' + target + '/deviceDetectedAt', now.toUTCString());
374         Property.setFlag('devices/' + target + '/deviceDetectedAt', 'ARCHIVE', true);
375       }
376       var length = Property.getProperty('devices/' + target + '/config/length');
377       var onTime = Property.getProperty('devices/' + target + '/config/onTime');
378       var latest = now.getTime() + (length - onTime) * 1000;
379       latest = new Date(latest);
380       delayOn(target, latest.toUTCString());
381     }
382   }
383   Property.store();
384   Property.setProperty('sendRequest', 'newConfig');
385 }
386 else {
387   if (data.message.indexOf('Could not find device with dsid') === 0) {
388     log('Token expired. Retrieving a new one', 0);
389     getJSONToken();
390   }
391   else {
392     log('Device' + dsid + 'not present', 5);
393     log('Message:' + data.message, 8);
394   }
395 }
396 }
397
398
399
400 //check if tcp connection still open
401 var time = Property.getProperty('timeLastDataReceived');
402 if (time !== null) {
403   var now = new Date();
404   var dt = Math.round((now.getTime() - time)/1000);
405   log('Heard nothing from server for ' + dt + 's', 10);
406   var pollInterval = Property.getProperty('pollInterval');
407   if (pollInterval === null) {
408     pollInterval = 60;
409   }
410   if (dt > pollInterval * 2) {
411     log('Heard nothing from server for ' + dt + 's', 3);
412     log('The connection probably died. Opening a new one...', 3);
413     persistentConnection();
414   }
415 }
416 else{
417   log('No timeLastDataReceived node. Opening a new connection...', 3);
418   persistentConnection();
419 }
420 }
421 } //poll
422
423 function raiseNextPollEvent() {
424   var pollInterval = Property.getProperty('pollInterval');
425   if (pollInterval === null) {
426     pollInterval = 60;
427   }
428   //register new poll event
429   var event = new TimedEvent('smart-grid', '+' + pollInterval, {action_type: 'poll'});
430   var pollEventId = event.raise();
431   Property.setProperty('pollEventId', pollEventId);
432 }
433
434 function updateConfig(config) {
435   Property.setProperty('serverAddress', config.server);
436   Property.setFlag('serverAddress', 'ARCHIVE', true);
437   Property.setProperty('pollInterval', config.poll);
438   Property.setFlag('pollInterval', 'ARCHIVE', true);

```

```

439     Property.setProperty('serverPort', config.port);
440     Property.setFlag('serverPort', 'ARCHIVE', true);
441     Property.store();
442
443     var pollEventId = Property.getProperty('pollEventId');
444     if (pollEventId !== null) {
445       Property.getNode('/system/EventInterpreter/ScheduledEvents').removeChild(
446         pollEventId);
447     }
448
449     var clockOK = Property.getProperty('clockDriftOK');
450     if (clockOK !== null && clockOK) {
451       Property.getNode('/scripts/smart-grid').removeChild('clockDriftOK');
452       Property.getNode('/scripts/smart-grid').removeChild('checkingClockDrift');
453     }
454     timeDriftCheck();
455   }
456
457   function shortOff(xmlDoc) {
458     var dsid = xmlDoc.documentElement.text;
459     var seconds = xmlDoc.documentElement.getAttribute('seconds');
460     var device = getDevices().byDSID(dsid);
461     var now = new Date();
462
463     // check that the device is actually configured as shortOff device
464     var deviceType = Property.getProperty('devices/' + dsid + '/config/type');
465     if (deviceType === null || deviceType !== 'off') {
466       log('Device' + dsid + 'is not configured shortOff', 0);
467       return true;
468     }
469
470     //check that it is not already in shortOff state
471     var shortOffProp = Property.getProperty('devices/' + dsid + '/shortOff');
472     if (shortOffProp === null && shortOffProp) {
473       if (seconds === '0') {
474         log('Turning on device' + dsid, 3);
475         device.turnOn();
476
477         var shortOffEventId = Property.getProperty('devices/' + dsid + '/shortOffEventId
478           ');
479         Property.getNode('/system/EventInterpreter/ScheduledEvents').removeChild(
480           shortOffEventId);
481
482         var lastStart = Property.getProperty('devices/' + dsid + '/actualShortOffStart')
483           ;
484         var time = new Date(lastStart);
485         delta = (now.getTime() - time.getTime()) / 1000;
486         delta = Math.round(delta);
487
488         var lastTime = Property.getProperty('devices/' + dsid + '/actualShortOffTime');
489
490         var shortOffSlotTime = Property.getProperty('devices/' + dsid + '/
491           shortOffSlotTime');
492         Property.setProperty('devices/' + dsid + '/shortOffSlotTime', (shortOffSlotTime
493           - lastTime + delta));
494         Property.setFlag('devices/' + dsid + '/shortOffSlotTime', 'ARCHIVE', true);
495
496         Property.setProperty('devices/' + dsid + '/shortOff', false);
497         Property.setFlag('devices/' + dsid + '/shortOff', 'ARCHIVE', true);
498
499         Property.store();
500       }
501     }
502     //check that we are allowed to turn off the device under the constraints of the
503     //configuration
504     var shortOffSlotLength = Property.getProperty('devices/' + dsid + '/config/
505       slotLength');
506     var shortOffMaxOff = Property.getProperty('devices/' + dsid + '/config/offTime');
507     if (shortOffSlotLength === null || shortOffMaxOff === null) {
508       log('Device' + dsid + 'is not configured properly', 0);
509       return true;
510     }
511
512     var shortOffSlotTime = null;
513     var shortOffSlotStart = Property.getProperty('devices/' + dsid + '/shortOffSlotStart
514       ');
515     if (shortOffSlotStart === null) {
516       shortOffSlotStart = now;

```



```

594     }
595
596     //check if the device is already on
597     var delayOnProp = Property.getProperty('devices/' + dsid + '/delayOn');
598     if (delayOnProp !== null && delayOnProp) {
599         log('Device' + dsid + 'is already delayOn', 0);
600         return;
601     }
602
603     var at = now;
604     if (atString !== '') {
605         var temp = new Date(atString);
606         //check that the given time is not in the past
607         if (now.getTime() < temp.getTime()) {
608             at = temp;
609         }
610     }
611
612     //check that the current slot is not yet done already
613     var slotStartTime = Property.getProperty('devices/' + dsid + '/config/startTime');
614     if (slotStartTime === null) {
615         log('The slot is already done', 0);
616         return;
617     }
618     slotStartTime = new Date(slotStartTime);
619
620     //check if the specified time is not too late
621     var slotLength = Property.getProperty('devices/' + dsid + '/config/length');
622     var onTime = Property.getProperty('devices/' + dsid + '/config/onTime');
623     var latestStart = slotStartTime.getTime() + (slotLength - onTime) * 1000;
624     if (at.getTime() > latestStart) {
625         latestStart = new Date(latestStart);
626         log('Time' + at.toUTCString() + 'is too late. Starting at' + latestStart.toUTCString(), 0);
627         at = latestStart;
628     }
629
630     //check that the slot already started at the time specified
631     if (at.getTime() < slotStartTime.getTime()) {
632         log('The slot did not start at the given time', 0);
633         return;
634     }
635
636     //get event id of the already scheduled event
637     var oldEvent = Property.getProperty('devices/' + dsid + '/delayOnEventId');
638     if (oldEvent !== null) {
639         //delete the event
640         Property.getNode('/system/EventInterpreter/ScheduledEvents').removeChild(oldEvent);
641         log('Rescheduling device' + dsid, 3);
642     }
643
644     var dt = at.getTime() - now.getTime();
645     dt = Math.floor(dt/1000);
646     log('Turning on device' + dsid + 'in' + dt + 's', 3);
647     if (dt <= 0) {
648         delayOnStart(dsid);
649     } else {
650         var event = new TimedEvent('smart-grid', '+' + dt, {action_type: 'delayOnStart', dsid: dsid});
651         var delayOnEventId = event.raise();
652
653         //not needed on crash
654         Property.setProperty('devices/' + dsid + '/delayOnEventId', delayOnEventId);
655
656         Property.setProperty('devices/' + dsid + '/delayOnPlannedTime', at.toUTCString());
657         ;
658         Property.setFlag('devices/' + dsid + '/delayOnPlannedTime', 'ARCHIVE', true);
659         Property.store();
660     }
661 }
662
663 function delayOnStart(dsid) {
664     log('Turning on device' + dsid, 3);
665     getDevices().byDSID(dsid).turnOn();
666
667     //schedule off event
668     var onTime = Property.getProperty('devices/' + dsid + '/config/onTime');
669     log('and turning it off in' + onTime + 's', 3);
670     var event = new TimedEvent('smart-grid', '+' + onTime, {action_type: 'delayOnReset', dsid: dsid});
671     var delayOnResetEventId = event.raise();
672

```

```

673     Property.setProperty('devices/' + dsid + '/delayOnResetEventId', delayOnResetEventId
674     );
675     //save everything to be able to recover after crash
676     Property.setProperty('devices/' + dsid + '/delayOn', true);
677     Property.setFlag('devices/' + dsid + '/delayOn', 'ARCHIVE', true);
678     Property.setProperty('devices/' + dsid + '/delayOnTime', (new Date()).toUTCString())
679     ;
680     Property.setFlag('devices/' + dsid + '/delayOnTime', 'ARCHIVE', true);
681     Property.setProperty('sendRequest', 'newConfig');
682     Property.store();
683   }
684 }
685
686 function delayOnReset(dsid) {
687   log('Resetting delay on device ' + dsid, 3);
688   getDevices().byDSID(dsid).turnOff();
689   Property.setProperty('devices/' + dsid + '/delayOn', false);
690   Property.setFlag('devices/' + dsid + '/delayOn', 'ARCHIVE', true);
691
692   Property.getNode('devices/' + dsid + '/config').removeChild('startTime');
693
694   Property.store();
695 }
696
697
698 function getMeterValues(delta) {
699   log('Sending meter values', 10);
700   var now = new Date();
701   now = now.getTime();
702   var dsms = Apartment.getDSMeters();
703
704   if (delta === null) {
705     delta = 10000;
706   } else {
707     delta = Math.ceil(delta / (1000 * 1)); //ms * logging interval
708   }
709   var result = [];
710   var min = delta;
711
712   var firstStamp = null;
713   var dataLength = 0;
714
715   if (dsms.length > 0) {
716     log('Reading meter values for dsm', + dsms[0].dsid, 10);
717     var dsmData = Metering.getValues(dsms[0].dsid, 'consumption', 1);
718     dataLength = dsmData.length;
719
720     if (dataLength > 0) {
721       min = Math.min(delta, dataLength);
722       firstStamp = dsmData[dataLength - 1].timestamp;
723       for (j = dataLength - 1; j >= dataLength - min; j--) {
724         var point = dsmData[j];
725         var date = new Date(point.timestamp.replace(/-/g, '/'));
726         if (date.getTime() <= now) {
727           result.push({timestamp: date.toUTCString(), value: point.value});
728         }
729         else {
730           log('Time too new', 1);
731         }
732       }
733       if (result.length !== min) {
734         log("min changed", 1);
735         min = result.length;
736     }
737
738     var i, j;
739     for (i = 1; i < dsms.length; i++) {
740       log('Reading meter values for dsm', + dsms[i].dsid, 10);
741       dsmData = Metering.getValues(dsms[i].dsid, 'consumption', 1);
742       dataLength = dsmData.length;
743       j = dataLength;
744       while (j > 0 && dsmData[j - 1].timestamp !== firstStamp) {
745         j--;
746       }
747       if (j === 0) {
748         log("other dsm data is newer", 1);
749         continue;
750       }
751       else if (j !== dataLength) {
752         log("newer value then on other dsm", 10);
753         var oldLength = dsmData.length;
754         var num = dataLength - j;
    }
  }
}

```

```

755         dsmData.splice(j, num);
756         dataLength = dsmData.length;
757         log('Removing ' + num + ' elements old:' + oldLength + ' new:' + dataLength,
758             10);
759     }
760
761     if (dataLength < min) {
762         log('Resizing result', 10);
763         result.splice(dataLength, min - dataLength);
764         min = dataLength;
765         log('New length of array:' + min, 10);
766     }
767
768     for (j = 0; j < min; j++) {
769         var point = dsmData[dataLength - 1 - j];
770         var date = new Date(point.timestamp.replace(/-/g, '/'));
771         if (date.toUTCString() === result[j].timestamp) {
772             result[j].value += point.value;
773         } else {
774             log('Timestamp does not match', 10);
775         }
776     }
777 }
778 }
779 log('Data length ' + dataLength, 19);
780 log('Result length ' + result.length, 19);
781 var xml = '<consumptions>' + objToXml(result) + '</consumptions>';
782 return xml;
783 }
784 }
785
786 function persistentConnection() {
787     var socket = new TcpSocket();
788     var myLastTime = null;
789     var lastMeterTime = null;
790
791     //tried to open a server tcp socket. But just one connection can be accepted and
792     //rebuilding the socket always failed.
793     //TODO: open bug for this issue.
794     /*var control = null;
795     function closeServerSocket(){
796         if(control !== null){
797             //control.close();
798             log('WORKS', 0);
799         } else{
800             log('control null', 0);
801         }
802     }
803     var clientS = null;
804     function receiveControl(){
805         clientS.receiveLine(1024, function(data){
806             log('data ' + data, 0);
807             //receiveControl();
808             clientS.close();
809             setTimeout(closeServerSocket, 1000);
810         }, '\n');
811     }
812     function connectionReceived(clientSocket){
813         log('Received connection', 0);
814         if(socket !== null){
815             log('Jupieeeeeeee', 0);
816         }
817         //clientSocket.close();
818         clientS = clientSocket;
819         receiveControl();
820         //clientSocket.send('asdfasdf');
821
822         //setTimeout(closeServerSocket, 10000);
823         //control.close();
824         //control.accept(connectionReceived);
825         //buildServerSocket();
826         log('Works', 0);
827         //control.accept(connectionReceived);
828
829     }
830     function buildServerSocket(){
831         control = new TcpSocket();
832         control.bind(50006, function(state){
833             if(state){
834                 control.accept(connectionReceived);
835             }
836             else{

```

```

837         log('NAK', 0);
838     }
839   });
840 }
841 buildServerSocket();*/
842
843 //add a listener to a property to be able to be woken up on a event and send stuff
844 //this does not work properly either -> going back to polling
845 /*var listenerId = Property.getProperty('listenerId');
846 if(listenerId !== null){
847   Property.removeListener(listenerId);
848 }
849 Property.setProperty('sendRequest', '');
850 listenerId = Property.setListener('sendRequest', function() {
851   var requestType = Property.getProperty('sendRequest');
852   if (requestType === 'newConfig') {
853     log('Sending new config', 3);
854     //the old receive gets somehow killed
855     socket.send(getConfigXML(), sent);
856   } else if (requestType === 'measurements') {
857     //log('Sending measurement values', 3);
858     socket.send(getMeterValues(), sent);
859   } else {
860     log('Unknown send request ' + requestType, 0);
861   }
862 });
863 Property.setProperty('listenerId', listenerId);*/
864
865 function updateLastReceivedTime() {
866   var now = new Date();
867   myLastTime = ""+now.getTime();
868   Property.setProperty('timeLastDataReceived', myLastTime);
869 }
870
871 function receive(){
872   var requestType = Property.getProperty('sendRequest');
873   if (requestType === 'newConfig') {
874     log('Sending new config', 3);
875     Property.setProperty('sendRequest', '');
876     //the old receive gets somehow killed
877     socket.send(getConfigXML(), sent);
878     return;
879   }
880   socket.receiveLine(1024, function(data) {
881     if (host !== Property.getProperty('serverAddress') || port !== Property.
882       getProperty('serverPort')) {
883       log('The address of the server changed. Closing connection', 3);
884       return;
885     }
886
887     if (Property.getProperty('timeLastDataReceived') === myLastTime) {
888       updateLastReceivedTime();
889       if (data === '') {
890         //error
891         log('Remote side closed connection', 3);
892         socket.close();
893       } else {
894         var xmlDoc = new REXML(data);
895         if (xmlDoc.rootElement.type === 'element') {
896           var name = xmlDoc.rootElement.name;
897           if (name === 'ping') {
898             var delta = null;
899             var now = new Date();
900             if (lastMeterTime !== null) {
901               delta = now.getTime() - lastMeterTime.getTime();
902             }
903             else {
904               delta = 10000000;
905             }
906             log(delta + '', 11);
907             lastMeterTime = now;
908             log('Received ping', 20);
909             socket.send(getMeterValues(delta), sent);
910           }
911           else if (name === 'shortoff') {
912             shortOff(xmlDoc);
913             receive();
914           }
915           else if (name === 'delayon') {
916             delayOn(xmlDoc.rootElement.text, xmlDoc.rootElement.attribute('at'));
917             receive();
918           }
919         }
920       }
921     }
922   }
923 }

```

```

920         else {
921             log(data, 3);
922             receive();
923         }
924     } else {
925         log(data, 3);
926         receive();
927     }
928 }
929 }
930 else {
931     log('Another tcp connection is open. I will close', 3);
932 }
933 },
934 '\n');
935 }
936 function sent(bytesSent) {
937     if (bytesSent > 0) {
938         log('Sent the message', 20);
939         receive();
940     } else {
941         log('Could not send the message', 0);
942     }
943 }
944 }
945
946 var host = Property.getProperty('serverAddress');
947 var port = Property.getProperty('serverPort');
948 if (host === null || port === null) {
949     log('Debug: Server not specified', 3);
950     return;
951 }
952 updateLastReceivedTime();
953 socket.connect(host, port, function(state) {
954     if (state) {
955         log('Connected', 10);
956         Property.setProperty('sendRequest', 'newConfig');
957         receive();
958     } else {
959         log('Connection failed', 0);
960     }
961 });
962 } //persistentConnection
963
964 function objToXml(obj) {
965     var rString = '';
966     if (typeof obj === 'object') {
967         if (obj.constructor.toString().indexOf('Array') !== -1) {
968             for (i = 0; i < obj.length; i++) {
969                 rString = rString + '<item>' + objToXml(obj[i]) + '</item>';
970             }
971         } else {
972             for (i in obj) {
973                 var val = objToXml(obj[i]);
974                 if (!val) {
975                     return false;
976                 }
977                 rString += '<' + i + '>' + val + '</' + i + '>';
978             }
979         }
980     }
981 }
982 else if (typeof obj === 'string') {
983     rString = obj;
984 }
985 else if (obj.toString) {
986     rString = obj.toString();
987 }
988 else {
989     return false;
990 }
991
992 return rString;
993 }
994
995 function getConfigXML() {
996     log('Entering getConfigXML', 20);
997     var devices = [];
998     getDevices().perform(function(device) {
999         var type = Property.getProperty('devices/' + device.dsid + '/config/type');
1000        var config = {};
1001        config.type = type;
1002        config.id = device.dsid;
1003        if (type === 'on') {

```

```

1004     config.startTime = Property.getProperty('devices/' + device.dsid + '/config/
1005         startTime');
1006     var delayOn = Property.getProperty('devices/' + device.dsid + '/delayOn');
1007     if (config.startTime !== null && (delayOn === null || !delayOn) ) {
1008         config.slotLength = Property.getProperty('devices/' + device.dsid + '/config/
1009             length');
1010         config.onTime = Property.getProperty('devices/' + device.dsid + '/config/
1011             onTime');
1012         devices.push(config);
1013     }
1014     else if (type === 'off') {
1015         config.slotLength = Property.getProperty('devices/' + device.dsid + '/config/
1016             slotLength');
1017         config.offTime = Property.getProperty('devices/' + device.dsid + '/config/
1018             offTime');
1019         devices.push(config);
1020     }
1021 });
1022
1023     var xml = '<config>' + objToXml(devices) + '</config>';
1024     log('Sending config to the server', 10);
1025     log('Sending following config to the server' + xml, 15);
1026
1027     return xml;
1028 }
1029
1030 function newConfig(dsid, config) {
1031     log('New config for ' + dsid, 4);
1032
1033     var configObject = JSON.parse(config);
1034
1035     var type = configObject.type;
1036     var oldType = Property.getProperty('devices/' + dsid + '/config/type');
1037
1038     //reset device
1039     if (oldType === 'off') {
1040         var isOff = Property.getProperty('devices/' + dsid + '/shortOff');
1041         if (isOff !== null && isOff) {
1042             var resetEventId = Property.getProperty('devices/' + dsid + '/shortOffEventId');
1043             Property.getNode('/system/EventInterpreter/ScheduledEvents').removeChild(
1044                 resetEventId);
1045
1046             //turn the device on again to save the freezer
1047             shortOffReset(dsid);
1048         }
1049     }
1050     else if (oldType === 'on') {
1051         //keep the on/off state of the device
1052         var on = Property.getProperty('devices/' + dsid + '/delayOn');
1053         if (on !== null && on) {
1054             var delayOnResetEventId = Property.getProperty('devices/' + dsid + '/
1055                 delayOnResetEventId');
1056             Property.getNode('/system/EventInterpreter/ScheduledEvents').removeChild(
1057                 delayOnResetEventId);
1058         }
1059     }
1060     else {
1061         var startTime = Property.getProperty('devices/' + dsid + '/config/startTime');
1062         if (startTime !== null) {
1063             var delayOnEventId = Property.getProperty('devices/' + dsid + '/delayOnEventId
1064                 ');
1065             Property.getNode('/system/EventInterpreter/ScheduledEvents').removeChild(
1066                 delayOnEventId);
1067         }
1068     }
1069 }
1070 var oldDetectionDevice = Property.getProperty('devices/' + dsid + '/config/
1071     onDetectionDevice');
1072 if (oldDetectionDevice !== null) {
1073     oldDetectionDevice.removeChild(dsid);
1074
1075     Property.setProperty('devices/' + dsid + '/config/type', type);
1076     Property.setFlag('devices/' + dsid + '/config/type', 'ARCHIVE', true);

```

```

1077
1078     Property.setProperty('devices/' + dsid + '/config/all', config);
1079     Property.setFlag('devices/' + dsid + '/config/all', 'ARCHIVE', true);
1080
1081     if (type === 'on') {
1082         var length = 60 * (configObject.lengthHours * 60 + configObject.lengthMinutes);
1083         Property.setProperty('devices/' + dsid + '/config/length', length);
1084         Property.setFlag('devices/' + dsid + '/config/length', 'ARCHIVE', true);
1085
1086         var onTime = 60 * (configObject.onHours * 60 + configObject.onMinutes);
1087         Property.setProperty('devices/' + dsid + '/config/onTime', onTime);
1088         Property.setFlag('devices/' + dsid + '/config/onTime', 'ARCHIVE', true);
1089
1090         log(configObject.startDetection, 10);
1091         if (configObject.startDetection === 'singleSlot') {
1092             var date = configObject.date.replace(/-/g, '/').split('T')[0] + ':' + configObject.startHours + ':' + configObject.startMinutes + ':0';
1093             var starttime = new Date(date);
1094             Property.setProperty('devices/' + dsid + '/config/startTime', starttime.toUTCString());
1095             Property.setFlag('devices/' + dsid + '/config/startTime', 'ARCHIVE', true);
1096
1097             var latest = starttime.getTime() + (length - onTime) * 1000;
1098             latest = new Date(latest);
1099             delayOn(dsid, latest.toUTCString());
1100         }
1101     else if (configObject.startDetection === 'device') {
1102         //this is not a new config we just have to detect a slot start
1103         var device = configObject.onDetectionDevice;
1104         log('Device detection with device' + device, 5);
1105         Property.setProperty('devicesToObserve/' + device + '/' + dsid, true);
1106         Property.setFlag('devicesToObserve/' + device + '/' + dsid, 'ARCHIVE', true);
1107
1108         Property.setProperty('devices/' + dsid + '/config/onDetectionDevice', device);
1109         Property.setFlag('devices/' + dsid + '/config/onDetectionDevice', 'ARCHIVE', true);
1110     }
1111 }
1112 else if (type === 'off') {
1113     var slotLength = 60 * (configObject.slotLengthHours * 60 + configObject.slotLengthMinutes);
1114     Property.setProperty('devices/' + dsid + '/config/slotLength', slotLength);
1115     Property.setFlag('devices/' + dsid + '/config/slotLength', 'ARCHIVE', true);
1116
1117     var offTime = 60 * (configObject.offTimeHours * 60 + configObject.offTimeMinutes);
1118     Property.setProperty('devices/' + dsid + '/config/offTime', offTime);
1119     Property.setFlag('devices/' + dsid + '/config/offTime', 'ARCHIVE', true);
1120 }
1121 log('New config saved for ' + dsid, 4);
1122 Property.setProperty('sendRequest', 'newConfig');

1123 Property.store();
1124 } //newConfig
1125
1126
1127 function main() {
1128     if (raisedEvent.name === 'running') {
1129         // Prepare app
1130         Property.load();
1131         LOG.login('');
1132         LOG.login('=====');
1133         LOG.login('# smart-grid running #');
1134         LOG.login('=====');
1135
1136         timeDriftCheck();
1137         return;
1138     }

1139     var action_type = raisedEvent.parameter.action_type;
1140     if (action_type === 'config') {
1141         log('Debug: Updating the server configuration', 5);
1142         log(raisedEvent.parameter.params, 5);
1143         updateConfig(JSON.parse(raisedEvent.parameter.params));
1144         return;
1145     }

1146
1147
1148     var clockOK = Property.getProperty('clockDriftOK');
1149     if (clockOK !== null && clockOK) {
1150         if (action_type === 'poll') {
1151             log('Debug: Doing a poll to the electricity provider', 20);
1152             poll();
1153             raiseNextPollEvent();
1154         }
1155     }
1156 else if (action_type === 'configDevice') {

```

```

1157     newConfig(raisedEvent.parameter.deviceId, raisedEvent.parameter.config);
1158 }
1159 else if (action_type === 'shortOffReset') {
1160   shortOffReset(raisedEvent.parameter.dsid);
1161 }
1162 else if (action_type === 'delayOnStart') {
1163   delayOnStart(raisedEvent.parameter.dsid);
1164 }
1165 else if (action_type === 'delayOnReset') {
1166   delayOnReset(raisedEvent.parameter.dsid);
1167 }
1168 else {
1169   log('Debug: Strange: received event not prepared for', 0);
1170   log('Debug: smart-grid main call with action-type:',
1171       + raisedEvent.parameter.action_type
1172       + (raisedEvent.parameter.params === undefined ? ''
1173          : ' and params:'
1174          + raisedEvent.parameter.params.toString()
1175          ), 0
1176 );
1177 }
1178 }
1179 else {
1180   log('Clock not verified to have the correct time. Waiting.', + action_type, 0);
1181 }
1182 } // main
1183
1184 main();

```

Listing B.2: scripts/smartgrid.js

B.1.3 User Interface

```

1 Ext.define('DSS.addon.SmartGrid.ConfigWindow', {
2   extend: 'Ext.window.Window',
3   title: 'SmartGrid',
4   layout: 'fit',
5   closeAction: 'hide',
6
7   constructor: function(config){
8     this.initConfig(config);
9     this.callParent(arguments);
10   },
11
12   /** Server Address field */
13   serverAddress: null,
14
15   /** Form Panel */
16   formPanel: null,
17
18   /** Boolean to remember if the data is already fetched */
19   fetchedData: false,
20
21   initComponent: function(){
22     var me = this;
23
24     Ext.define('configPanel', {
25       extend: 'Ext.form.Panel',
26       bodyPadding: 5, // Don't want content to crunch against the borders
27       width: 300,
28       items: [
29         {
30           name: 'serverAddress',
31           fieldLabel: _("Server address"),
32           xtype: 'textfield',
33           allowBlank: false
34         },
35         {
36           name: 'serverPort',
37           fieldLabel: _("Server port"),
38           xtype: 'numberfield',
39           allowBlank: false,
40           minValue: 0,
41           maxValue: 65535,
42
43           // Remove spinner buttons, and arrow key and mouse wheel listeners
44           hideTrigger: true,
45           keyNavEnabled: false,
46
47         }
48       ]
49     });
50
51     this.items.push(configPanel);
52   }
53 }
54
55 ConfigWindow.superclass.constructor.call(this, config);
56
57 this.show();
58
59 }
60
61 ConfigWindow.prototype.show = function() {
62   this.callParent(arguments);
63
64   var me = this;
65
66   Ext.create('Ext.data.Store', {
67     id: 'gridData',
68     autoLoad: true,
69     proxy: {
70       type: 'ajax',
71       url: 'http://127.0.0.1:8080/api/v1/devices'
72     }
73   }).load({
74     success: function(store, operation) {
75       me.fetchedData = true;
76       me.serverAddress = store.getAt(0).get('ip');
77       me.serverPort = store.getAt(0).get('port');
78     }
79   });
80
81   me.setButtons([
82     {
83       text: 'OK',
84       handler: function() {
85         me.close();
86       }
87     }
88   ]);
89
90   me.show();
91 }
92
93 ConfigWindow.prototype.onClose = function() {
94   this.callParent();
95
96   var me = this;
97
98   Ext.create('Ext.data.Store', {
99     id: 'gridData',
100    autoLoad: true,
101    proxy: {
102      type: 'ajax',
103      url: 'http://127.0.0.1:8080/api/v1/devices'
104    }
105  }).load({
106    success: function(store, operation) {
107      me.fetchedData = true;
108      me.serverAddress = store.getAt(0).get('ip');
109      me.serverPort = store.getAt(0).get('port');
110    }
111  });
112
113  me.setButtons([
114    {
115      text: 'OK',
116      handler: function() {
117        me.close();
118      }
119    }
120  ]);
121
122  me.show();
123 }
124
125 ConfigWindow.prototype.onClose = function() {
126   this.callParent();
127
128   var me = this;
129
130   Ext.create('Ext.data.Store', {
131     id: 'gridData',
132     autoLoad: true,
133     proxy: {
134       type: 'ajax',
135       url: 'http://127.0.0.1:8080/api/v1/devices'
136     }
137   }).load({
138     success: function(store, operation) {
139       me.fetchedData = true;
140       me.serverAddress = store.getAt(0).get('ip');
141       me.serverPort = store.getAt(0).get('port');
142     }
143   });
144
145   me.setButtons([
146     {
147       text: 'OK',
148       handler: function() {
149         me.close();
150       }
151     }
152   ]);
153
154   me.show();
155 }
156
157 ConfigWindow.prototype.onClose = function() {
158   this.callParent();
159
160   var me = this;
161
162   Ext.create('Ext.data.Store', {
163     id: 'gridData',
164     autoLoad: true,
165     proxy: {
166       type: 'ajax',
167       url: 'http://127.0.0.1:8080/api/v1/devices'
168     }
169   }).load({
170     success: function(store, operation) {
171       me.fetchedData = true;
172       me.serverAddress = store.getAt(0).get('ip');
173       me.serverPort = store.getAt(0).get('port');
174     }
175   });
176
177   me.setButtons([
178     {
179       text: 'OK',
180       handler: function() {
181         me.close();
182       }
183     }
184   ]);
185
186   me.show();
187 }
188
189 ConfigWindow.prototype.onClose = function() {
190   this.callParent();
191
192   var me = this;
193
194   Ext.create('Ext.data.Store', {
195     id: 'gridData',
196     autoLoad: true,
197     proxy: {
198       type: 'ajax',
199       url: 'http://127.0.0.1:8080/api/v1/devices'
200     }
201   }).load({
202     success: function(store, operation) {
203       me.fetchedData = true;
204       me.serverAddress = store.getAt(0).get('ip');
205       me.serverPort = store.getAt(0).get('port');
206     }
207   });
208
209   me.setButtons([
210     {
211       text: 'OK',
212       handler: function() {
213         me.close();
214       }
215     }
216   ]);
217
218   me.show();
219 }
220
221 ConfigWindow.prototype.onClose = function() {
222   this.callParent();
223
224   var me = this;
225
226   Ext.create('Ext.data.Store', {
227     id: 'gridData',
228     autoLoad: true,
229     proxy: {
230       type: 'ajax',
231       url: 'http://127.0.0.1:8080/api/v1/devices'
232     }
233   }).load({
234     success: function(store, operation) {
235       me.fetchedData = true;
236       me.serverAddress = store.getAt(0).get('ip');
237       me.serverPort = store.getAt(0).get('port');
238     }
239   });
240
241   me.setButtons([
242     {
243       text: 'OK',
244       handler: function() {
245         me.close();
246       }
247     }
248   ]);
249
250   me.show();
251 }
252
253 ConfigWindow.prototype.onClose = function() {
254   this.callParent();
255
256   var me = this;
257
258   Ext.create('Ext.data.Store', {
259     id: 'gridData',
260     autoLoad: true,
261     proxy: {
262       type: 'ajax',
263       url: 'http://127.0.0.1:8080/api/v1/devices'
264     }
265   }).load({
266     success: function(store, operation) {
267       me.fetchedData = true;
268       me.serverAddress = store.getAt(0).get('ip');
269       me.serverPort = store.getAt(0).get('port');
270     }
271   });
272
273   me.setButtons([
274     {
275       text: 'OK',
276       handler: function() {
277         me.close();
278       }
279     }
280   ]);
281
282   me.show();
283 }
284
285 ConfigWindow.prototype.onClose = function() {
286   this.callParent();
287
288   var me = this;
289
290   Ext.create('Ext.data.Store', {
291     id: 'gridData',
292     autoLoad: true,
293     proxy: {
294       type: 'ajax',
295       url: 'http://127.0.0.1:8080/api/v1/devices'
296     }
297   }).load({
298     success: function(store, operation) {
299       me.fetchedData = true;
300       me.serverAddress = store.getAt(0).get('ip');
301       me.serverPort = store.getAt(0).get('port');
302     }
303   });
304
305   me.setButtons([
306     {
307       text: 'OK',
308       handler: function() {
309         me.close();
310       }
311     }
312   ]);
313
314   me.show();
315 }
316
317 ConfigWindow.prototype.onClose = function() {
318   this.callParent();
319
320   var me = this;
321
322   Ext.create('Ext.data.Store', {
323     id: 'gridData',
324     autoLoad: true,
325     proxy: {
326       type: 'ajax',
327       url: 'http://127.0.0.1:8080/api/v1/devices'
328     }
329   }).load({
330     success: function(store, operation) {
331       me.fetchedData = true;
332       me.serverAddress = store.getAt(0).get('ip');
333       me.serverPort = store.getAt(0).get('port');
334     }
335   });
336
337   me.setButtons([
338     {
339       text: 'OK',
340       handler: function() {
341         me.close();
342       }
343     }
344   ]);
345
346   me.show();
347 }
348
349 ConfigWindow.prototype.onClose = function() {
350   this.callParent();
351
352   var me = this;
353
354   Ext.create('Ext.data.Store', {
355     id: 'gridData',
356     autoLoad: true,
357     proxy: {
358       type: 'ajax',
359       url: 'http://127.0.0.1:8080/api/v1/devices'
360     }
361   }).load({
362     success: function(store, operation) {
363       me.fetchedData = true;
364       me.serverAddress = store.getAt(0).get('ip');
365       me.serverPort = store.getAt(0).get('port');
366     }
367   });
368
369   me.setButtons([
370     {
371       text: 'OK',
372       handler: function() {
373         me.close();
374       }
375     }
376   ]);
377
378   me.show();
379 }
380
381 ConfigWindow.prototype.onClose = function() {
382   this.callParent();
383
384   var me = this;
385
386   Ext.create('Ext.data.Store', {
387     id: 'gridData',
388     autoLoad: true,
389     proxy: {
390       type: 'ajax',
391       url: 'http://127.0.0.1:8080/api/v1/devices'
392     }
393   }).load({
394     success: function(store, operation) {
395       me.fetchedData = true;
396       me.serverAddress = store.getAt(0).get('ip');
397       me.serverPort = store.getAt(0).get('port');
398     }
399   });
400
401   me.setButtons([
402     {
403       text: 'OK',
404       handler: function() {
405         me.close();
406       }
407     }
408   ]);
409
410   me.show();
411 }
412
413 ConfigWindow.prototype.onClose = function() {
414   this.callParent();
415
416   var me = this;
417
418   Ext.create('Ext.data.Store', {
419     id: 'gridData',
420     autoLoad: true,
421     proxy: {
422       type: 'ajax',
423       url: 'http://127.0.0.1:8080/api/v1/devices'
424     }
425   }).load({
426     success: function(store, operation) {
427       me.fetchedData = true;
428       me.serverAddress = store.getAt(0).get('ip');
429       me.serverPort = store.getAt(0).get('port');
430     }
431   });
432
433   me.setButtons([
434     {
435       text: 'OK',
436       handler: function() {
437         me.close();
438       }
439     }
440   ]);
441
442   me.show();
443 }
444
445 ConfigWindow.prototype.onClose = function() {
446   this.callParent();
447
448   var me = this;
449
450   Ext.create('Ext.data.Store', {
451     id: 'gridData',
452     autoLoad: true,
453     proxy: {
454       type: 'ajax',
455       url: 'http://127.0.0.1:8080/api/v1/devices'
456     }
457   }).load({
458     success: function(store, operation) {
459       me.fetchedData = true;
460       me.serverAddress = store.getAt(0).get('ip');
461       me.serverPort = store.getAt(0).get('port');
462     }
463   });
464
465   me.setButtons([
466     {
467       text: 'OK',
468       handler: function() {
469         me.close();
470       }
471     }
472   ]);
473
474   me.show();
475 }
476
477 ConfigWindow.prototype.onClose = function() {
478   this.callParent();
479
480   var me = this;
481
482   Ext.create('Ext.data.Store', {
483     id: 'gridData',
484     autoLoad: true,
485     proxy: {
486       type: 'ajax',
487       url: 'http://127.0.0.1:8080/api/v1/devices'
488     }
489   }).load({
490     success: function(store, operation) {
491       me.fetchedData = true;
492       me.serverAddress = store.getAt(0).get('ip');
493       me.serverPort = store.getAt(0).get('port');
494     }
495   });
496
497   me.setButtons([
498     {
499       text: 'OK',
500       handler: function() {
501         me.close();
502       }
503     }
504   ]);
505
506   me.show();
507 }
508
509 ConfigWindow.prototype.onClose = function() {
510   this.callParent();
511
512   var me = this;
513
514   Ext.create('Ext.data.Store', {
515     id: 'gridData',
516     autoLoad: true,
517     proxy: {
518       type: 'ajax',
519       url: 'http://127.0.0.1:8080/api/v1/devices'
520     }
521   }).load({
522     success: function(store, operation) {
523       me.fetchedData = true;
524       me.serverAddress = store.getAt(0).get('ip');
525       me.serverPort = store.getAt(0).get('port');
526     }
527   });
528
529   me.setButtons([
530     {
531       text: 'OK',
532       handler: function() {
533         me.close();
534       }
535     }
536   ]);
537
538   me.show();
539 }
540
541 ConfigWindow.prototype.onClose = function() {
542   this.callParent();
543
544   var me = this;
545
546   Ext.create('Ext.data.Store', {
547     id: 'gridData',
548     autoLoad: true,
549     proxy: {
550       type: 'ajax',
551       url: 'http://127.0.0.1:8080/api/v1/devices'
552     }
553   }).load({
554     success: function(store, operation) {
555       me.fetchedData = true;
556       me.serverAddress = store.getAt(0).get('ip');
557       me.serverPort = store.getAt(0).get('port');
558     }
559   });
560
561   me.setButtons([
562     {
563       text: 'OK',
564       handler: function() {
565         me.close();
566       }
567     }
568   ]);
569
570   me.show();
571 }
572
573 ConfigWindow.prototype.onClose = function() {
574   this.callParent();
575
576   var me = this;
577
578   Ext.create('Ext.data.Store', {
579     id: 'gridData',
580     autoLoad: true,
581     proxy: {
582       type: 'ajax',
583       url: 'http://127.0.0.1:8080/api/v1/devices'
584     }
585   }).load({
586     success: function(store, operation) {
587       me.fetchedData = true;
588       me.serverAddress = store.getAt(0).get('ip');
589       me.serverPort = store.getAt(0).get('port');
590     }
591   });
592
593   me.setButtons([
594     {
595       text: 'OK',
596       handler: function() {
597         me.close();
598       }
599     }
600   ]);
601
602   me.show();
603 }
604
605 ConfigWindow.prototype.onClose = function() {
606   this.callParent();
607
608   var me = this;
609
610   Ext.create('Ext.data.Store', {
611     id: 'gridData',
612     autoLoad: true,
613     proxy: {
614       type: 'ajax',
615       url: 'http://127.0.0.1:8080/api/v1/devices'
616     }
617   }).load({
618     success: function(store, operation) {
619       me.fetchedData = true;
620       me.serverAddress = store.getAt(0).get('ip');
621       me.serverPort = store.getAt(0).get('port');
622     }
623   });
624
625   me.setButtons([
626     {
627       text: 'OK',
628       handler: function() {
629         me.close();
630       }
631     }
632   ]);
633
634   me.show();
635 }
636
637 ConfigWindow.prototype.onClose = function() {
638   this.callParent();
639
640   var me = this;
641
642   Ext.create('Ext.data.Store', {
643     id: 'gridData',
644     autoLoad: true,
645     proxy: {
646       type: 'ajax',
647       url: 'http://127.0.0.1:8080/api/v1/devices'
648     }
649   }).load({
650     success: function(store, operation) {
651       me.fetchedData = true;
652       me.serverAddress = store.getAt(0).get('ip');
653       me.serverPort = store.getAt(0).get('port');
654     }
655   });
656
657   me.setButtons([
658     {
659       text: 'OK',
660       handler: function() {
661         me.close();
662       }
663     }
664   ]);
665
666   me.show();
667 }
668
669 ConfigWindow.prototype.onClose = function() {
670   this.callParent();
671
672   var me = this;
673
674   Ext.create('Ext.data.Store', {
675     id: 'gridData',
676     autoLoad: true,
677     proxy: {
678       type: 'ajax',
679       url: 'http://127.0.0.1:8080/api/v1/devices'
680     }
681   }).load({
682     success: function(store, operation) {
683       me.fetchedData = true;
684       me.serverAddress = store.getAt(0).get('ip');
685       me.serverPort = store.getAt(0).get('port');
686     }
687   });
688
689   me.setButtons([
690     {
691       text: 'OK',
692       handler: function() {
693         me.close();
694       }
695     }
696   ]);
697
698   me.show();
699 }
700
701 ConfigWindow.prototype.onClose = function() {
702   this.callParent();
703
704   var me = this;
705
706   Ext.create('Ext.data.Store', {
707     id: 'gridData',
708     autoLoad: true,
709     proxy: {
710       type: 'ajax',
711       url: 'http://127.0.0.1:8080/api/v1/devices'
712     }
713   }).load({
714     success: function(store, operation) {
715       me.fetchedData = true;
716       me.serverAddress = store.getAt(0).get('ip');
717       me.serverPort = store.getAt(0).get('port');
718     }
719   });
720
721   me.setButtons([
722     {
723       text: 'OK',
724       handler: function() {
725         me.close();
726       }
727     }
728   ]);
729
730   me.show();
731 }
732
733 ConfigWindow.prototype.onClose = function() {
734   this.callParent();
735
736   var me = this;
737
738   Ext.create('Ext.data.Store', {
739     id: 'gridData',
740     autoLoad: true,
741     proxy: {
742       type: 'ajax',
743       url: 'http://127.0.0.1:8080/api/v1/devices'
744     }
745   }).load({
746     success: function(store, operation) {
747       me.fetchedData = true;
748       me.serverAddress = store.getAt(0).get('ip');
749       me.serverPort = store.getAt(0).get('port');
750     }
751   });
752
753   me.setButtons([
754     {
755       text: 'OK',
756       handler: function() {
757         me.close();
758       }
759     }
760   ]);
761
762   me.show();
763 }
764
765 ConfigWindow.prototype.onClose = function() {
766   this.callParent();
767
768   var me = this;
769
770   Ext.create('Ext.data.Store', {
771     id: 'gridData',
772     autoLoad: true,
773     proxy: {
774       type: 'ajax',
775       url: 'http://127.0.0.1:8080/api/v1/devices'
776     }
777   }).load({
778     success: function(store, operation) {
779       me.fetchedData = true;
780       me.serverAddress = store.getAt(0).get('ip');
781       me.serverPort = store.getAt(0).get('port');
782     }
783   });
784
785   me.setButtons([
786     {
787       text: 'OK',
788       handler: function() {
789         me.close();
790       }
791     }
792   ]);
793
794   me.show();
795 }
796
797 ConfigWindow.prototype.onClose = function() {
798   this.callParent();
799
800   var me = this;
801
802   Ext.create('Ext.data.Store', {
803     id: 'gridData',
804     autoLoad: true,
805     proxy: {
806       type: 'ajax',
807       url: 'http://127.0.0.1:8080/api/v1/devices'
808     }
809   }).load({
810     success: function(store, operation) {
811       me.fetchedData = true;
812       me.serverAddress = store.getAt(0).get('ip');
813       me.serverPort = store.getAt(0).get('port');
814     }
815   });
816
817   me.setButtons([
818     {
819       text: 'OK',
820       handler: function() {
821         me.close();
822       }
823     }
824   ]);
825
826   me.show();
827 }
828
829 ConfigWindow.prototype.onClose = function() {
830   this.callParent();
831
832   var me = this;
833
834   Ext.create('Ext.data.Store', {
835     id: 'gridData',
836     autoLoad: true,
837     proxy: {
838       type: 'ajax',
839       url: 'http://127.0.0.1:8080/api/v1/devices'
840     }
841   }).load({
842     success: function(store, operation) {
843       me.fetchedData = true;
844       me.serverAddress = store.getAt(0).get('ip');
845       me.serverPort = store.getAt(0).get('port');
846     }
847   });
848
849   me.setButtons([
850     {
851       text: 'OK',
852       handler: function() {
853         me.close();
854       }
855     }
856   ]);
857
858   me.show();
859 }
860
861 ConfigWindow.prototype.onClose = function() {
862   this.callParent();
863
864   var me = this;
865
866   Ext.create('Ext.data.Store', {
867     id: 'gridData',
868     autoLoad: true,
869     proxy: {
870       type: 'ajax',
871       url: 'http://127.0.0.1:8080/api/v1/devices'
872     }
873   }).load({
874     success: function(store, operation) {
875       me.fetchedData = true;
876       me.serverAddress = store.getAt(0).get('ip');
877       me.serverPort = store.getAt(0).get('port');
878     }
879   });
880
881   me.setButtons([
882     {
883       text: 'OK',
884       handler: function() {
885         me.close();
886       }
887     }
888   ]);
889
890   me.show();
891 }
892
893 ConfigWindow.prototype.onClose = function() {
894   this.callParent();
895
896   var me = this;
897
898   Ext.create('Ext.data.Store', {
899     id: 'gridData',
900     autoLoad: true,
901     proxy: {
902       type: 'ajax',
903       url: 'http://127.0.0.1:8080/api/v1/devices'
904     }
905   }).load({
906     success: function(store, operation) {
907       me.fetchedData = true;
908       me.serverAddress = store.getAt(0).get('ip');
909       me.serverPort = store.getAt(0).get('port');
910     }
911   });
912
913   me.setButtons([
914     {
915       text: 'OK',
916       handler: function() {
917         me.close();
918       }
919     }
920   ]);
921
922   me.show();
923 }
924
925 ConfigWindow.prototype.onClose = function() {
926   this.callParent();
927
928   var me = this;
929
930   Ext.create('Ext.data.Store', {
931     id: 'gridData',
932     autoLoad: true,
933     proxy: {
934       type: 'ajax',
935       url: 'http://127.0.0.1:8080/api/v1/devices'
936     }
937   }).load({
938     success: function(store, operation) {
939       me.fetchedData = true;
940       me.serverAddress = store.getAt(0).get('ip');
941       me.serverPort = store.getAt(0).get('port');
942     }
943   });
944
945   me.setButtons([
946     {
947       text: 'OK',
948       handler: function() {
949         me.close();
950       }
951     }
952   ]);
953
954   me.show();
955 }
956
957 ConfigWindow.prototype.onClose = function() {
958   this.callParent();
959
960   var me = this;
961
962   Ext.create('Ext.data.Store', {
963     id: 'gridData',
964     autoLoad: true,
965     proxy: {
966       type: 'ajax',
967       url: 'http://127.0.0.1:8080/api/v1/devices'
968     }
969   }).load({
970     success: function(store, operation) {
971       me.fetchedData = true;
972       me.serverAddress = store.getAt(0).get('ip');
973       me.serverPort = store.getAt(0).get('port');
974     }
975   });
976
977   me.setButtons([
978     {
979       text: 'OK',
980       handler: function() {
981         me.close();
982       }
983     }
984   ]);
985
986   me.show();
987 }
988
989 ConfigWindow.prototype.onClose = function() {
990   this.callParent();
991
992   var me = this;
993
994   Ext.create('Ext.data.Store', {
995     id: 'gridData',
996     autoLoad: true,
997     proxy: {
998       type: 'ajax',
999       url: 'http://127.0.0.1:8080/api/v1/devices'
1000      }
1001    }).load({
1002      success: function(store, operation) {
1003        me.fetchedData = true;
1004        me.serverAddress = store.getAt(0).get('ip');
1005        me.serverPort = store.getAt(0).get('port');
1006      }
1007    });
1008
1009    me.setButtons([
1010      {
1011        text: 'OK',
1012        handler: function() {
1013          me.close();
1014        }
1015      }
1016    ]);
1017
1018    me.show();
1019 }
1020
1021 ConfigWindow.prototype.onClose = function() {
1022   this.callParent();
1023
1024   var me = this;
1025
1026   Ext.create('Ext.data.Store', {
1027     id: 'gridData',
1028     autoLoad: true,
1029     proxy: {
1030       type: 'ajax',
1031       url: 'http://127.0.0.1:8080/api/v1/devices'
1032     }
1033   }).load({
1034     success: function(store, operation) {
1035       me.fetchedData = true;
1036       me.serverAddress = store.getAt(0).get('ip');
1037       me.serverPort = store.getAt(0).get('port');
1038     }
1039   });
1040
1041   me.setButtons([
1042     {
1043       text: 'OK',
1044       handler: function() {
1045         me.close();
1046       }
1047     }
1048   ]);
1049
1050   me.show();
1051 }
1052
1053 ConfigWindow.prototype.onClose = function() {
1054   this.callParent();
1055
1056   var me = this;
1057
1058   Ext.create('Ext.data.Store', {
1059     id: 'gridData',
1060     autoLoad: true,
1061     proxy: {
1062       type: 'ajax',
1063       url: 'http://127.0.0.1:8080/api/v1/devices'
1064     }
1065   }).load({
1066     success: function(store, operation) {
1067       me.fetchedData = true;
1068       me.serverAddress = store.getAt(0).get('ip');
1069       me.serverPort = store.getAt(0).get('port');
1070     }
1071   });
1072
1073   me.setButtons([
1074     {
1075       text: 'OK',
1076       handler: function() {
1077         me.close();
1078       }
1079     }
1080   ]);
1081
1082   me.show();
1083 }
1084
1085 ConfigWindow.prototype.onClose = function() {
1086   this.callParent();
1087
1088   var me = this;
1089
1090   Ext.create('Ext.data.Store', {
1091     id: 'gridData',
1092     autoLoad: true,
1093     proxy: {
1094       type: 'ajax',
1095       url: 'http://127.0.0.1:8080/api/v1/devices'
1096     }
1097   }).load({
1098     success: function(store, operation) {
1099       me.fetchedData = true;
1100       me.serverAddress = store.getAt(0).get('ip');
1101       me.serverPort = store.getAt(0).get('port');
1102     }
1103   });
1104
1105   me.setButtons([
1106     {
1107       text: 'OK',
1108       handler: function() {
1109         me.close();
1110       }
1111     }
1112   ]);
1113
1114   me.show();
1115 }
1116
1117 ConfigWindow.prototype.onClose = function() {
1118   this.callParent();
1119
1120   var me = this;
1121
1122   Ext.create('Ext.data.Store', {
1123     id: 'gridData',
1124     autoLoad: true,
1125     proxy: {
1126       type: 'ajax',
1127       url: 'http://127.0.0.1:8080/api/v1/devices'
1128     }
1129   }).load({
1130     success: function(store, operation) {
1131       me.fetchedData = true;
1132       me.serverAddress = store.getAt(0).get('ip');
1133       me.serverPort = store.getAt(0).get('port');
1134     }
1135   });
1136
1137   me.setButtons([
1138     {
1139       text: 'OK',
1140       handler: function() {
1141         me.close();
1142       }
1143     }
1144   ]);
1145
1146   me.show();
1147 }
1148
1149 ConfigWindow.prototype.onClose = function() {
1150   this.callParent();
1151
1152   var me = this;
1153
1154   Ext.create('Ext.data.Store', {
1155     id: 'gridData',
1156     autoLoad: true,
1157     proxy: {
1158       type: 'ajax',
1159       url: 'http://127.0.0.1:8080/api/v1/devices'
1160     }
1161   }).load({
1162     success: function(store, operation) {
1163       me.fetchedData = true;
1164       me.serverAddress = store.getAt(0).get('ip');
1165       me.serverPort = store.getAt(0).get('port');
1166     }
1167   });
1168
1169   me.setButtons([
1170     {
1171       text: 'OK',
1172       handler: function() {
1173         me.close();
1174       }
1175     }
1176   ]);
1177
1178   me.show();
1179 }
1180
1181 ConfigWindow.prototype.onClose = function() {
1182   this.callParent();
1183
1184   var me = this;
1185
1186   Ext.create('Ext.data.Store', {
1187     id: 'gridData',
1188     autoLoad: true,
1189     proxy: {
1190       type: 'ajax',
1191       url: 'http://127.0.0.1:8080/api/v1/devices'
1192     }
1193   }).load({
1194     success: function(store, operation) {
1195       me.fetchedData = true;
1196       me.serverAddress = store.getAt(0).get('ip');
1197       me.serverPort = store.getAt(0).get('port');
1198     }
1199   });
1200
1201   me.setButtons([
1202     {
1203       text: 'OK',
1204       handler: function() {
1205         me.close();
1206       }
1207     }
1208   ]);
1209
1210   me.show();
1211 }
1212
1213 ConfigWindow.prototype.onClose = function() {
1214   this.callParent();
1215
1216   var me = this;
1217
1218   Ext.create('Ext.data.Store', {
1219     id: 'gridData',
1220     autoLoad: true,
1221     proxy: {
1222       type: 'ajax',
1223       url: 'http://127.0.0.1:8080/api/v1/devices'
1224     }
1225   }).load({
1226     success: function(store, operation) {
1227       me.fetchedData = true;
1228       me.serverAddress = store.getAt(0).get('ip');
1229       me.serverPort = store.getAt(0).get('port');
1230     }
1231   });
1232
1233   me.setButtons([
1234     {
1235       text: 'OK',
1236       handler: function() {
1237         me.close();
1238       }
1239     }
1240   ]);
1241
1242   me.show();
1243 }
1244
1245 ConfigWindow.prototype.onClose = function() {
1246   this.callParent();
1247
1248   var me = this;
1249
1250   Ext.create('Ext.data.Store', {
1251     id: 'gridData',
1252     autoLoad: true,
1253     proxy: {
12
```

```

46 mouseWheelEnabled: false
47 },
48 name: 'pollInterval',
49 fieldLabel: _('Poll Interval'),
50 xtype: 'numberfield',
51 minValue: 10, //prevents lower values than 10
52 allowBlank: false,
53
54 // Remove spinner buttons, and arrow key and mouse wheel listeners
55 hideTrigger: true,
56 keyNavEnabled: false,
57 mouseWheelEnabled: false
58 }
59 ],
60
61 /** footer bar */
62 fbar: {
63 items: [
64 {
65 text: _('Cancel'),
66 id: 'btn-cancel'
67 },
68 {
69 text: _('Save'),
70 id: 'btn-save'
71 }
72 ]
73 },
74
75 constructor: function(config){
76 this.initConfig(config);
77 this.callParent(arguments);
78 },
79
80 initComponent: function(){
81 var me = this;
82 me.addEvents({
83 eventhide: true
84 });
85
86 me.callParent(arguments);
87 me.initPage();
88 },
89
90
91 initPage: function() {
92 var me = this;
93
94 Ext.getCmp('btn-save').handler = function() {
95 var form = me.getForm();
96 if (form.isValid()) {
97 me.saveIt(form);
98 }
99 };
100 Ext.getCmp('btn-cancel').handler = function() {
101 me.fireEvent('eventhide');
102 };
103 },
104
105 saveIt: function(form){
106 var me = this;
107
108 var data = form.getFieldValues();
109 var serverAddress = me.down('[name=serverAddress]');
110 var pollInterval = me.down('[name=pollInterval]');
111 var serverPort = me.down('[name=serverPort]');
112 serverAddress.resetOriginalValue();
113 pollInterval.resetOriginalValue();
114 serverPort.resetOriginalValue();
115
116
117 var params = {
118 poll: data.pollInterval,
119 server: data.serverAddress,
120 port: data.serverPort
121 };
122
123 var event = Ext.create('DSS.json.Event', {name: 'smart-grid'});
124 event.raise({
125 actionType: 'config',
126 params: Ext.JSON.encode(params)
127 }, {
128 success: function(){
129 me.fireEvent('eventhide');

```

```

130         },
131         failure: function() {
132             Ext.Msg.alert(‘Error’, ‘Couldn’t create timed event on server’));
133         });
134     });
135 },
136
137     getField: function(path, success){
138         var me = this;
139         Ext.Ajax.request({
140             disableCaching: true,
141             method: ‘GET’,
142             timeout: 20000,
143             url: ‘/json/’ + path,
144             success: function(response){
145                 var data = Ext.JSON.decode(response.responseText);
146                 if(data.ok){
147                     success(data.result.value);
148                 }
149                 else{
150                     me.enable();
151                 }
152             },
153             failure: function(){
154                 me.enable();
155             }
156         });
157     },
158
159     beforeShow: function(){
160         var me = this;
161         if(!me.fetchData){
162             me.disable();
163             me.fetchData = true;
164             me.getField(“property/getInteger?path=/scripts/smart-grid/pollInterval”,
165                         function(data){
166                             var pollInterval = me.down(‘[name=pollInterval]’);
167                             pollInterval.setRawValue(data);
168                             pollInterval.resetOriginalValue();
169                             me.getField(“property/getString?path=/scripts/smart-grid/serverAddress”,
170                                         function(data){
171                                             var serverAddress = me.down(‘[name=serverAddress]’);
172                                             serverAddress.setRawValue(data);
173                                             serverAddress.resetOriginalValue();
174                                             me.getField(“property/getInteger?path=/scripts/smart-grid/serverPort”,
175                                                 function(data){
176                                                     var serverPort = me.down(‘[name=serverPort]’);
177                                                     serverPort.setRawValue(data);
178                                                     serverPort.resetOriginalValue();
179                                                     me.enable();
180                                                 });
181                                         });
182                                         else{
183                                             me.getForm().reset();
184                                         }
185                                         });
186                                         );
187                                         formPanel = Ext.create(‘configPanel’, { property: me.property });
188                                         me.items = formPanel;
189                                         me.items.on{
190                                             eventhide: function(){
191                                                 me.hide();
192                                             }
193                                         );
194                                         me.addListener(‘beforeshow’, function(){
195                                             formPanel.beforeShow();
196                                         });
197                                         me.callParent(arguments);
198                                         }
199                                         );
200                                         );
201                                         );

```

Listing B.3: ui/js/configWindow.js

```

1 Ext.define(‘DSS.addon.SmartGrid.DeviceWindow’, {
2     extend: ‘Ext.window.Window’,
3     title: ‘SmartGrid’,
4     layout: ‘fit’,

```

```

5   closeAction: 'hide',
6
7  /** The store object is received with the config parameter in the constructor */
8  store: null,
9
10 /* currently edited device */
11 device: null,
12
13 constructor: function(config){
14   this.initConfig(config);
15   this.callParent(arguments);
16 },
17
18 items: [
19   {
20     xtype: 'form',
21     id: 'form',
22     bodyPadding: 5,
23     items: [
24       {
25         boxLabel: _('Excluded'),
26         name: 'type',
27         inputValue: 'exclude',
28         id: 'exclude',
29         xtype: 'radiofield',
30         width: 130
31       },
32       {
33         xtype: 'container',
34         layout: 'column',
35         items: [
36           {
37             boxLabel: _('DelayedON'),
38             name: 'type',
39             inputValue: 'on',
40             id: 'on',
41             xtype: 'radiofield',
42             width: 130
43           },
44           {
45             xtype: 'container',
46             id: 'onContainer',
47             items: [
48               {
49                 xtype: 'fieldcontainer',
50                 fieldLabel: _('Slotstartdetection'),
51                 labelWidth: 120,
52                 items: [
53                   {
54                     xtype: 'container',
55                     layout: 'column',
56                     items: [
57                       {
58                         boxLabel: _('Singleslot'),
59                         id: 'onSingleSlot',
60                         name: 'startDetection',
61                         inputValue: 'singleSlot',
62                         xtype: 'radiofield'
63                       },
64                       {
65                         xtype: 'container',
66                         id: 'startTimeContainer',
67                         layout: 'column',
68                         items: [
69                           {
70                             id: 'onSingleSlotDate',
71                             name: 'date',
72                             xtype: 'datefield',
73                             margin: '0 0 0 5',
74                             allowBlank: false
75                           },
76                           {
77                             id: 'onSingleSlotTime',
78                             name: 'start',
79                             xtype: 'dssTimeSelection'
80                           }
81                         ]
82                       }
83                     ]
84                   },
85                   {
86                     xtype: 'container',
87                     layout: 'column',
88                     id: 'startDetectionContainer',

```

```

89         items: [
90             {
91                 boxLabel: _("On\u00addevice\u00adpresence"),
92                 id: 'onDetection',
93                 name: 'startDetection',
94                 inputValue: 'device',
95                 xtype: 'radiofield',
96                 margin: '0\u005c0\u005c0'
97             }
98         ]
99     }
100 }
101 }
102 {
103     xtype: 'fieldcontainer',
104     labelWidth: 120,
105     fieldLabel: _("Slot\u00adlength"),
106     items: [
107         {
108             id: 'onSlotLength',
109             name: 'length',
110             xtype: 'dssTimeSelection',
111             type: 'length'
112         }
113     ]
114 }
115 {
116     xtype: 'fieldcontainer',
117     labelWidth: 120,
118     fieldLabel: _("ON\u00adtime"),
119     items: [
120         {
121             id: 'onOnTime',
122             name: 'on',
123             xtype: 'dssTimeSelection',
124             type: 'length'
125         }
126     ]
127 }
128 /*
129 */
130 {
131     xtype: 'fieldcontainer',
132     fieldLabel: _("Interruption allowed"),
133     labelWidth: 120,
134     items: [
135         {
136             boxLabel: _("Yes"),
137             id: 'interruptTrue',
138             name: 'interrupt',
139             inputValue: "yes",
140             xtype: 'radiofield',
141         },
142         {
143             boxLabel: _("No, device needs to run in a row"),
144             id: 'interruptFalse',
145             name: 'interrupt',
146             inputValue: "no",
147             xtype: 'radiofield',
148         }
149     ]
150 }
151 }
152 {
153     xtype: 'container',
154     layout: 'column',
155     items: [
156         {
157             boxLabel: _("Short\u00adPeriod\u00adOFF"),
158             name: 'type',
159             inputValue: 'off',
160             id: 'off',
161             xtype: 'radiofield',
162             width: 130
163         },
164         {
165             xtype: 'container',
166             id: 'offContainer',
167             items: [
168                 {
169                     xtype: 'fieldcontainer',
170                     fieldLabel: _("Slot\u00adlength"),
171                     labelWidth: 120,
172

```

```

173         items: [
174             {
175                 id: 'offSlotLength',
176                 name: 'slotLength',
177                 xtype: 'dssTimeSelection',
178                 type: 'length'
179             }
180         ]
181     },
182     {
183         xtype: 'fieldcontainer',
184         fieldLabel: _('OFF_Utime'),
185         labelWidth: 120,
186         items: [
187             {
188                 id: 'offOffTime',
189                 name: 'offTime',
190                 xtype: 'dssTimeSelection',
191                 type: 'length'
192             }
193         ]
194     }
195 ],
196 ],
197 ],
198 ],
199 buttons: [
200     {
201         text: _("Cancel"),
202         id: 'btn-cancel',
203     },
204     {
205         text: _("Save"),
206         id: 'btn-save',
207     }
208 ],
209 ],
210 ],
211 ],
212 ],
213 initComponent: function(){
214     var me = this;
215     me.callParent(arguments);
216     me.initPage();
217 },
218 },
219 enableOff: function(enable){
220     Ext.getCmp('offOffTime').enable(enable);
221     Ext.getCmp('offSlotLength').enable(enable);
222 },
223 },
224 enableOn: function(enable){
225     Ext.getCmp('onSlotLength').enable(enable);
226     Ext.getCmp('onOnTime').enable(enable);
227     if(enable){
228         Ext.getCmp('onSingleSlot').enable();
229         Ext.getCmp('onDetection').enable();
230         Ext.getCmp('onDetection').setValue(false);
231         Ext.getCmp('onDetection').setValue(true);
232     }
233     else{
234         Ext.getCmp('onSingleSlot').disable();
235         Ext.getCmp('onSingleSlotDate').disable();
236         Ext.getCmp('onDetectionDevice').disable();
237         Ext.getCmp('onDetection').disable();
238         Ext.getCmp('onSingleSlotTime').enable(false);
239     }
240 },
241 },
242 initPage: function(){
243     var me = this;
244     Ext.getCmp('btn-cancel').handler = function() {
245         me.hide();
246     };
247     Ext.getCmp('btn-save').handler = function() {
248         var form = Ext.getCmp('form').getForm();
249         if(form.isValid()){
250             var type = '';
251             if(Ext.getCmp('on').getValue()){
252                 var length = Ext.getCmp('onSlotLength').getValue();
253                 var onTime = Ext.getCmp('onOnTime').getValue();
254                 type = 'on';
255                 if(onTime > length){
256

```

```

257         Ext.Msg.alert(_("Error"), _("ON time has to be smaller than the slot length"));
258         return;
259     }
260 }
261 else if(Ext.getCmp('off').getValue()){
262     var offTime = Ext.getCmp('offOffTime').getValue();
263     var length = Ext.getCmp('offSlotLength').getValue();
264     type = 'off';
265     if(offTime > length){
266         Ext.Msg.alert(_("Error"), _("OFF time has to be smaller than the slot length"));
267         return;
268     }
269 }
270
271 var data = form.getFieldValues();
272 data = Ext.JSON.encode(data);
273 me.setLoading(true);
274 var event = Ext.create('DSS.json.Event', {name: 'smart-grid'});
275 event.raise(
276 {
277     action_type: 'configDevice',
278     deviceId: me.device.get('id'),
279     config: data
280 },
281 {
282     success: function(){
283         var model = me.store.getById(me.device.get('id'));
284         model.set('smartGridType', type);
285         model.commit();
286         me.setLoading(false);
287         me.hide();
288     },
289     failure: function(){
290         Ext.Msg.alert(_("Error"), _("Couldn't send event to DSS"));
291         me.setLoading(false);
292         me.hide();
293     }
294 },
295 );
296 );
297 };
298 Ext.getCmp('exclude').handler = function(){
299     if(Ext.getCmp('exclude').getValue()){
300         me.enableOff(false);
301         me.enableOn(false);
302     }
303 };
304 Ext.getCmp('on').handler = function(){
305     if(Ext.getCmp('on').getValue()){
306         me.enableOff(false);
307         me.enableOn(true);
308     }
309 };
310 Ext.getCmp('onSingleSlot').handler = function(){
311     if(Ext.getCmp('onSingleSlot').getValue()){
312         Ext.getCmp('onSingleSlotDate').enable();
313         Ext.getCmp('onSingleSlotTime').enable(true);
314     }
315     else{
316         Ext.getCmp('onSingleSlotDate').disable();
317         Ext.getCmp('onSingleSlotTime').enable(false);
318     }
319 };
320 Ext.getCmp('onDetection').handler = function(){
321     if(Ext.getCmp('onDetection').getValue()){
322         Ext.getCmp('onDetectionDevice').enable();
323     }
324     else{
325         Ext.getCmp('onDetectionDevice').disable();
326     }
327 };
328 Ext.getCmp('off').handler = function(){
329     if(Ext.getCmp('off').getValue()){
330         me.enableOff(true);
331         me.enableOn(false);
332     }
333 };
334
335 var iconTpl = Ext.create('Ext.Template', [
336     // The pics are 16x16, +5 padding = 21
337     '<div style="',
338     '<tpl if="icon">',

```

```

339         'background:left_center_no-repeat_url(\`images/dss/{icon}\`);',
340         '</tpl>',
341         'min-height:16;',
342         'padding-left:21px;',
343         '<tpl_if="isPresent === false">color:gray;</tpl>',
344         ">{text}</div>',
345     ],
346   );
347   var combo = Ext.create('Ext.form.field.ComboBox', {
348     fieldLabel: '',
349     name: 'onDetectionDevice',
350     id: 'onDetectionDevice',
351     editable: false,
352     store: me.store,
353     queryMode: 'local',
354     displayField: 'name',
355     valueField: 'id',
356     listConfig: { itemTpl: iconTpl },
357     forceSelection: true
358   });
359   combo.on('render', function(thisBox) {
360     // also create and render the picker on box rendering
361     // otherwise render-time selection is not available
362     var picker = thisBox.getPicker();
363     picker.doAutoRender();
364   });
365   combo.on('select', function(field, value, options) {
366     //display the icon
367     var bg = 'background:none';
368     var icon = (value.length > 0 ? value[0].get('icon') : null);
369     if (icon) {
370       var url = 'images/dss/' + icon;
371       bg = "background:left_center_no-repeat_url('"+ url +"');";
372     }
373     field.setStyle(bg + 'padding-left:21px');
374   });
375
376   Ext.getCmp('startDetectionContainer').add(combo);
377 },
378
379 ajax: function(path, success){
380   var me = this;
381   Ext.Ajax.request({
382     disableCaching: true,
383     method: 'GET',
384     timeout: 20000,
385     url: '/json/' + path,
386     success: success,
387     failure: function(){
388       Ext.Msg.alert(_("A_erreur_happened"));
389       me.setLoading(false);
390       me.hide();
391     }
392   });
393 },
394 },
395
396 openDevice: function(device){
397   var me = this;
398   me.device = device;
399   me.setTitle(device.get('name'));
400   me.show();
401   me.setLoading(true);
402   me.ajax("property/getString?path=/scripts/smart-grid/devices/" + device.get('id')
403           + "/config/all", function(response){
404     var data = Ext.JSON.decode(response.responseText);
405     Ext.getCmp('onDetection').setValue(false);
406     Ext.getCmp('onDetection').setValue(true);
407     //Ext.getCmp('interruptTrue').setValue(true);
408     Ext.getCmp('exclude').setValue(false);
409     Ext.getCmp('exclude').setValue(true);
410     if(data.ok){
411       data = Ext.JSON.decode(data.result.value);
412       Ext.getCmp('form').getForm().setValues(data);
413       if(data.date !== undefined){
414         var dt = new Date(data.date);
415         Ext.getCmp('onSingleSlotDate').setValue(dt);
416       }
417       if(data.onDetectionDevice !== undefined){
418         var box = Ext.getCmp('onDetectionDevice');
419         box.fireEvent('select', box, [me.store.findRecord('id', data.
420           onDetectionDevice)], null);
421       }
422     }
423   })
424 }

```

```

421     me.setLoading(false);
422   });
423 }
424 });

```

Listing B.4: ui/js/deviceWindow.js

B.2 Demonstration Energy Provider

```

1 package ch.ethz.baumachr.energyProvider;
2
3 import java.io.StringReader;
4 import java.nio.channels.SocketChannel;
5 import java.text.ParseException;
6 import java.util.Date;
7 import java.util.HashMap;
8 import java.util.HashSet;
9 import java.util.Iterator;
10 import java.util.TreeSet;
11
12 import javax.xml.parsers.DocumentBuilder;
13 import javax.xml.parsers.DocumentBuilderFactory;
14 import javax.xml.parsers.ParserConfigurationException;
15
16 import org.w3c.dom.Document;
17 import org.w3c.dom.Node;
18 import org.w3c.dom.NodeList;
19 import org.xml.sax.InputSource;
20
21 public class Client {
22   private final StringBuilder stringBuilder;
23   private final DocumentBuilder domBuilder;
24   private final HashMap<String, Device> devices;
25   private final SocketChannel channel;
26   private final DSSServer server;
27
28   private final Logger LOG;
29
30   public Client(DSSServer server, SocketChannel channel) throws
31     ParserConfigurationException{
32     this.channel = channel;
33     this.server = server;
34
35     stringBuilder = new StringBuilder();
36     devices = new HashMap<String, Device>();
37
38     DocumentBuilderFactory f = DocumentBuilderFactory.newInstance();
39     domBuilder = f.newDocumentBuilder();
40
41     LOG = SingletonUtil.instance().LOG;
42   }
43
44   public void newData(byte[] data){
45     stringBuilder.append(new String(data));
46     removeCommands();
47   }
48
49   private void newConfig(Document dom){
50     boolean somethingChanged = false;
51     LOG.log("", 5);
52     LOG.log("New_config:", 5);
53     NodeList devicesNodeList = dom.getElementsByTagName("item");
54     HashSet<String> newIds = new HashSet<String>();
55     for(int i=0; i<devicesNodeList.getLength(); i++){
56       HashMap<String, String> params = new HashMap<String, String>();
57       NodeList paramNodes = devicesNodeList.item(i).getChildNodes();
58       for(int j=0; j<paramNodes.getLength(); j++){
59         Node param = paramNodes.item(j);
60         params.put(param.getNodeName(), param.getTextContent());
61       }
62       if(params.containsKey("id")){
63         String id = params.get("id");
64         newIds.add(id);
65         if(devices.containsKey(id)){
66           Device old = devices.get(id);
67           if(old.equals(params)){
68             continue;
69           }
70           devices.remove(id);
71           devices.put(id, params);
72         }
73       }
74     }
75   }
76
77   public void removeCommands(){
78     stringBuilder.replace(0, stringBuilder.length(),
79       "<?xml version='1.0' encoding='UTF-8'?><commands>" +
80       "</commands>");
81   }
82
83   public void sendCommand(String command, String id, String value){
84     String message = command + " " + id + " " + value;
85     try {
86       channel.write(ByteBuffer.wrap(message.getBytes()));
87     } catch (IOException e) {
88     }
89   }
90
91   public void close() throws IOException{
92     channel.close();
93   }
94
95   public void shutdown() throws IOException{
96     LOG.info("Shutting down client");
97     close();
98     server.shutdown();
99   }
100
101   public void start() throws IOException{
102     LOG.info("Starting client");
103     server.start();
104   }
105
106   public void stop() throws IOException{
107     LOG.info("Stopping client");
108     server.stop();
109   }
110
111   public void disconnect() throws IOException{
112     LOG.info("Disconnecting client");
113     channel.close();
114   }
115
116   public void connect() throws IOException{
117     LOG.info("Connecting client");
118     channel.connect();
119   }
120
121   public void disconnectAll() throws IOException{
122     LOG.info("Disconnecting all clients");
123     for(SocketChannel c : server.getChannels()) {
124       c.close();
125     }
126   }
127
128   public void connectAll() throws IOException{
129     LOG.info("Connecting all clients");
130     for(SocketChannel c : server.getChannels()) {
131       c.connect();
132     }
133   }
134
135   public void shutdownAll() throws IOException{
136     LOG.info("Shutting down all clients");
137     for(SocketChannel c : server.getChannels()) {
138       c.close();
139     }
140   }
141
142   public void startAll() throws IOException{
143     LOG.info("Starting all clients");
144     for(SocketChannel c : server.getChannels()) {
145       c.connect();
146     }
147   }
148
149   public void stopAll() throws IOException{
150     LOG.info("Stopping all clients");
151     for(SocketChannel c : server.getChannels()) {
152       c.close();
153     }
154   }
155
156   public void disconnectAll() throws IOException{
157     LOG.info("Disconnecting all clients");
158     for(SocketChannel c : server.getChannels()) {
159       c.close();
160     }
161   }
162
163   public void connectAll() throws IOException{
164     LOG.info("Connecting all clients");
165     for(SocketChannel c : server.getChannels()) {
166       c.connect();
167     }
168   }
169
170   public void shutdownAll() throws IOException{
171     LOG.info("Shutting down all clients");
172     for(SocketChannel c : server.getChannels()) {
173       c.close();
174     }
175   }
176
177   public void startAll() throws IOException{
178     LOG.info("Starting all clients");
179     for(SocketChannel c : server.getChannels()) {
180       c.connect();
181     }
182   }
183
184   public void stopAll() throws IOException{
185     LOG.info("Stopping all clients");
186     for(SocketChannel c : server.getChannels()) {
187       c.close();
188     }
189   }
190
191   public void disconnectAll() throws IOException{
192     LOG.info("Disconnecting all clients");
193     for(SocketChannel c : server.getChannels()) {
194       c.close();
195     }
196   }
197
198   public void connectAll() throws IOException{
199     LOG.info("Connecting all clients");
200     for(SocketChannel c : server.getChannels()) {
201       c.connect();
202     }
203   }
204
205   public void shutdownAll() throws IOException{
206     LOG.info("Shutting down all clients");
207     for(SocketChannel c : server.getChannels()) {
208       c.close();
209     }
210   }
211
212   public void startAll() throws IOException{
213     LOG.info("Starting all clients");
214     for(SocketChannel c : server.getChannels()) {
215       c.connect();
216     }
217   }
218
219   public void stopAll() throws IOException{
220     LOG.info("Stopping all clients");
221     for(SocketChannel c : server.getChannels()) {
222       c.close();
223     }
224   }
225
226   public void disconnectAll() throws IOException{
227     LOG.info("Disconnecting all clients");
228     for(SocketChannel c : server.getChannels()) {
229       c.close();
230     }
231   }
232
233   public void connectAll() throws IOException{
234     LOG.info("Connecting all clients");
235     for(SocketChannel c : server.getChannels()) {
236       c.connect();
237     }
238   }
239
240   public void shutdownAll() throws IOException{
241     LOG.info("Shutting down all clients");
242     for(SocketChannel c : server.getChannels()) {
243       c.close();
244     }
245   }
246
247   public void startAll() throws IOException{
248     LOG.info("Starting all clients");
249     for(SocketChannel c : server.getChannels()) {
250       c.connect();
251     }
252   }
253
254   public void stopAll() throws IOException{
255     LOG.info("Stopping all clients");
256     for(SocketChannel c : server.getChannels()) {
257       c.close();
258     }
259   }
260
261   public void disconnectAll() throws IOException{
262     LOG.info("Disconnecting all clients");
263     for(SocketChannel c : server.getChannels()) {
264       c.close();
265     }
266   }
267
268   public void connectAll() throws IOException{
269     LOG.info("Connecting all clients");
270     for(SocketChannel c : server.getChannels()) {
271       c.connect();
272     }
273   }
274
275   public void shutdownAll() throws IOException{
276     LOG.info("Shutting down all clients");
277     for(SocketChannel c : server.getChannels()) {
278       c.close();
279     }
280   }
281
282   public void startAll() throws IOException{
283     LOG.info("Starting all clients");
284     for(SocketChannel c : server.getChannels()) {
285       c.connect();
286     }
287   }
288
289   public void stopAll() throws IOException{
290     LOG.info("Stopping all clients");
291     for(SocketChannel c : server.getChannels()) {
292       c.close();
293     }
294   }
295
296   public void disconnectAll() throws IOException{
297     LOG.info("Disconnecting all clients");
298     for(SocketChannel c : server.getChannels()) {
299       c.close();
300     }
301   }
302
303   public void connectAll() throws IOException{
304     LOG.info("Connecting all clients");
305     for(SocketChannel c : server.getChannels()) {
306       c.connect();
307     }
308   }
309
310   public void shutdownAll() throws IOException{
311     LOG.info("Shutting down all clients");
312     for(SocketChannel c : server.getChannels()) {
313       c.close();
314     }
315   }
316
317   public void startAll() throws IOException{
318     LOG.info("Starting all clients");
319     for(SocketChannel c : server.getChannels()) {
320       c.connect();
321     }
322   }
323
324   public void stopAll() throws IOException{
325     LOG.info("Stopping all clients");
326     for(SocketChannel c : server.getChannels()) {
327       c.close();
328     }
329   }
330
331   public void disconnectAll() throws IOException{
332     LOG.info("Disconnecting all clients");
333     for(SocketChannel c : server.getChannels()) {
334       c.close();
335     }
336   }
337
338   public void connectAll() throws IOException{
339     LOG.info("Connecting all clients");
340     for(SocketChannel c : server.getChannels()) {
341       c.connect();
342     }
343   }
344
345   public void shutdownAll() throws IOException{
346     LOG.info("Shutting down all clients");
347     for(SocketChannel c : server.getChannels()) {
348       c.close();
349     }
350   }
351
352   public void startAll() throws IOException{
353     LOG.info("Starting all clients");
354     for(SocketChannel c : server.getChannels()) {
355       c.connect();
356     }
357   }
358
359   public void stopAll() throws IOException{
360     LOG.info("Stopping all clients");
361     for(SocketChannel c : server.getChannels()) {
362       c.close();
363     }
364   }
365
366   public void disconnectAll() throws IOException{
367     LOG.info("Disconnecting all clients");
368     for(SocketChannel c : server.getChannels()) {
369       c.close();
370     }
371   }
372
373   public void connectAll() throws IOException{
374     LOG.info("Connecting all clients");
375     for(SocketChannel c : server.getChannels()) {
376       c.connect();
377     }
378   }
379
380   public void shutdownAll() throws IOException{
381     LOG.info("Shutting down all clients");
382     for(SocketChannel c : server.getChannels()) {
383       c.close();
384     }
385   }
386
387   public void startAll() throws IOException{
388     LOG.info("Starting all clients");
389     for(SocketChannel c : server.getChannels()) {
390       c.connect();
391     }
392   }
393
394   public void stopAll() throws IOException{
395     LOG.info("Stopping all clients");
396     for(SocketChannel c : server.getChannels()) {
397       c.close();
398     }
399   }
400
401   public void disconnectAll() throws IOException{
402     LOG.info("Disconnecting all clients");
403     for(SocketChannel c : server.getChannels()) {
404       c.close();
405     }
406   }
407
408   public void connectAll() throws IOException{
409     LOG.info("Connecting all clients");
410     for(SocketChannel c : server.getChannels()) {
411       c.connect();
412     }
413   }
414
415   public void shutdownAll() throws IOException{
416     LOG.info("Shutting down all clients");
417     for(SocketChannel c : server.getChannels()) {
418       c.close();
419     }
420   }
421
422   public void startAll() throws IOException{
423     LOG.info("Starting all clients");
424     for(SocketChannel c : server.getChannels()) {
425       c.connect();
426     }
427   }
428
429   public void stopAll() throws IOException{
430     LOG.info("Stopping all clients");
431     for(SocketChannel c : server.getChannels()) {
432       c.close();
433     }
434   }
435
436   public void disconnectAll() throws IOException{
437     LOG.info("Disconnecting all clients");
438     for(SocketChannel c : server.getChannels()) {
439       c.close();
440     }
441   }
442
443   public void connectAll() throws IOException{
444     LOG.info("Connecting all clients");
445     for(SocketChannel c : server.getChannels()) {
446       c.connect();
447     }
448   }
449
450   public void shutdownAll() throws IOException{
451     LOG.info("Shutting down all clients");
452     for(SocketChannel c : server.getChannels()) {
453       c.close();
454     }
455   }
456
457   public void startAll() throws IOException{
458     LOG.info("Starting all clients");
459     for(SocketChannel c : server.getChannels()) {
460       c.connect();
461     }
462   }
463
464   public void stopAll() throws IOException{
465     LOG.info("Stopping all clients");
466     for(SocketChannel c : server.getChannels()) {
467       c.close();
468     }
469   }
470
471   public void disconnectAll() throws IOException{
472     LOG.info("Disconnecting all clients");
473     for(SocketChannel c : server.getChannels()) {
474       c.close();
475     }
476   }
477
478   public void connectAll() throws IOException{
479     LOG.info("Connecting all clients");
480     for(SocketChannel c : server.getChannels()) {
481       c.connect();
482     }
483   }
484
485   public void shutdownAll() throws IOException{
486     LOG.info("Shutting down all clients");
487     for(SocketChannel c : server.getChannels()) {
488       c.close();
489     }
490   }
491
492   public void startAll() throws IOException{
493     LOG.info("Starting all clients");
494     for(SocketChannel c : server.getChannels()) {
495       c.connect();
496     }
497   }
498
499   public void stopAll() throws IOException{
500     LOG.info("Stopping all clients");
501     for(SocketChannel c : server.getChannels()) {
502       c.close();
503     }
504   }
505
506   public void disconnectAll() throws IOException{
507     LOG.info("Disconnecting all clients");
508     for(SocketChannel c : server.getChannels()) {
509       c.close();
510     }
511   }
512
513   public void connectAll() throws IOException{
514     LOG.info("Connecting all clients");
515     for(SocketChannel c : server.getChannels()) {
516       c.connect();
517     }
518   }
519
520   public void shutdownAll() throws IOException{
521     LOG.info("Shutting down all clients");
522     for(SocketChannel c : server.getChannels()) {
523       c.close();
524     }
525   }
526
527   public void startAll() throws IOException{
528     LOG.info("Starting all clients");
529     for(SocketChannel c : server.getChannels()) {
530       c.connect();
531     }
532   }
533
534   public void stopAll() throws IOException{
535     LOG.info("Stopping all clients");
536     for(SocketChannel c : server.getChannels()) {
537       c.close();
538     }
539   }
540
541   public void disconnectAll() throws IOException{
542     LOG.info("Disconnecting all clients");
543     for(SocketChannel c : server.getChannels()) {
544       c.close();
545     }
546   }
547
548   public void connectAll() throws IOException{
549     LOG.info("Connecting all clients");
550     for(SocketChannel c : server.getChannels()) {
551       c.connect();
552     }
553   }
554
555   public void shutdownAll() throws IOException{
556     LOG.info("Shutting down all clients");
557     for(SocketChannel c : server.getChannels()) {
558       c.close();
559     }
560   }
561
562   public void startAll() throws IOException{
563     LOG.info("Starting all clients");
564     for(SocketChannel c : server.getChannels()) {
565       c.connect();
566     }
567   }
568
569   public void stopAll() throws IOException{
570     LOG.info("Stopping all clients");
571     for(SocketChannel c : server.getChannels()) {
572       c.close();
573     }
574   }
575
576   public void disconnectAll() throws IOException{
577     LOG.info("Disconnecting all clients");
578     for(SocketChannel c : server.getChannels()) {
579       c.close();
580     }
581   }
582
583   public void connectAll() throws IOException{
584     LOG.info("Connecting all clients");
585     for(SocketChannel c : server.getChannels()) {
586       c.connect();
587     }
588   }
589
590   public void shutdownAll() throws IOException{
591     LOG.info("Shutting down all clients");
592     for(SocketChannel c : server.getChannels()) {
593       c.close();
594     }
595   }
596
597   public void startAll() throws IOException{
598     LOG.info("Starting all clients");
599     for(SocketChannel c : server.getChannels()) {
600       c.connect();
601     }
602   }
603
604   public void stopAll() throws IOException{
605     LOG.info("Stopping all clients");
606     for(SocketChannel c : server.getChannels()) {
607       c.close();
608     }
609   }
610
611   public void disconnectAll() throws IOException{
612     LOG.info("Disconnecting all clients");
613     for(SocketChannel c : server.getChannels()) {
614       c.close();
615     }
616   }
617
618   public void connectAll() throws IOException{
619     LOG.info("Connecting all clients");
620     for(SocketChannel c : server.getChannels()) {
621       c.connect();
622     }
623   }
624
625   public void shutdownAll() throws IOException{
626     LOG.info("Shutting down all clients");
627     for(SocketChannel c : server.getChannels()) {
628       c.close();
629     }
630   }
631
632   public void startAll() throws IOException{
633     LOG.info("Starting all clients");
634     for(SocketChannel c : server.getChannels()) {
635       c.connect();
636     }
637   }
638
639   public void stopAll() throws IOException{
640     LOG.info("Stopping all clients");
641     for(SocketChannel c : server.getChannels()) {
642       c.close();
643     }
644   }
645
646   public void disconnectAll() throws IOException{
647     LOG.info("Disconnecting all clients");
648     for(SocketChannel c : server.getChannels()) {
649       c.close();
650     }
651   }
652
653   public void connectAll() throws IOException{
654     LOG.info("Connecting all clients");
655     for(SocketChannel c : server.getChannels()) {
656       c.connect();
657     }
658   }
659
660   public void shutdownAll() throws IOException{
661     LOG.info("Shutting down all clients");
662     for(SocketChannel c : server.getChannels()) {
663       c.close();
664     }
665   }
666
667   public void startAll() throws IOException{
668     LOG.info("Starting all clients");
669     for(SocketChannel c : server.getChannels()) {
670       c.connect();
671     }
672   }
673
674   public void stopAll() throws IOException{
675     LOG.info("Stopping all clients");
676     for(SocketChannel c : server.getChannels()) {
677       c.close();
678     }
679   }
680
681   public void disconnectAll() throws IOException{
682     LOG.info("Disconnecting all clients");
683     for(SocketChannel c : server.getChannels()) {
684       c.close();
685     }
686   }
687
688   public void connectAll() throws IOException{
689     LOG.info("Connecting all clients");
690     for(SocketChannel c : server.getChannels()) {
691       c.connect();
692     }
693   }
694
695   public void shutdownAll() throws IOException{
696     LOG.info("Shutting down all clients");
697     for(SocketChannel c : server.getChannels()) {
698       c.close();
699     }
700   }
701
702   public void startAll() throws IOException{
703     LOG.info("Starting all clients");
704     for(SocketChannel c : server.getChannels()) {
705       c.connect();
706     }
707   }
708
709   public void stopAll() throws IOException{
710     LOG.info("Stopping all clients");
711     for(SocketChannel c : server.getChannels()) {
712       c.close();
713     }
714   }
715
716   public void disconnectAll() throws IOException{
717     LOG.info("Disconnecting all clients");
718     for(SocketChannel c : server.getChannels()) {
719       c.close();
720     }
721   }
722
723   public void connectAll() throws IOException{
724     LOG.info("Connecting all clients");
725     for(SocketChannel c : server.getChannels()) {
726       c.connect();
727     }
728   }
729
730   public void shutdownAll() throws IOException{
731     LOG.info("Shutting down all clients");
732     for(SocketChannel c : server.getChannels()) {
733       c.close();
734     }
735   }
736
737   public void startAll() throws IOException{
738     LOG.info("Starting all clients");
739     for(SocketChannel c : server.getChannels()) {
740       c.connect();
741     }
742   }
743
744   public void stopAll() throws IOException{
745     LOG.info("Stopping all clients");
746     for(SocketChannel c : server.getChannels()) {
747       c.close();
748     }
749   }
750
751   public void disconnectAll() throws IOException{
752     LOG.info("Disconnecting all clients");
753     for(SocketChannel c : server.getChannels()) {
754       c.close();
755     }
756   }
757
758   public void connectAll() throws IOException{
759     LOG.info("Connecting all clients");
760     for(SocketChannel c : server.getChannels()) {
761       c.connect();
762     }
763   }
764
765   public void shutdownAll() throws IOException{
766     LOG.info("Shutting down all clients");
767     for(SocketChannel c : server.getChannels()) {
768       c.close();
769     }
770   }
771
772   public void startAll() throws IOException{
773     LOG.info("Starting all clients");
774     for(SocketChannel c : server.getChannels()) {
775       c.connect();
776     }
777   }
778
779   public void stopAll() throws IOException{
780     LOG.info("Stopping all clients");
781     for(SocketChannel c : server.getChannels()) {
782       c.close();
783     }
784   }
785
786   public void disconnectAll() throws IOException{
787     LOG.info("Disconnecting all clients");
788     for(SocketChannel c : server.getChannels()) {
789       c.close();
790     }
791   }
792
793   public void connectAll() throws IOException{
794     LOG.info("Connecting all clients");
795     for(SocketChannel c : server.getChannels()) {
796       c.connect();
797     }
798   }
799
800   public void shutdownAll() throws IOException{
801     LOG.info("Shutting down all clients");
802     for(SocketChannel c : server.getChannels()) {
803       c.close();
804     }
805   }
806
807   public void startAll() throws IOException{
808     LOG.info("Starting all clients");
809     for(SocketChannel c : server.getChannels()) {
810       c.connect();
811     }
812   }
813
814   public void stopAll() throws IOException{
815     LOG.info("Stopping all clients");
816     for(SocketChannel c : server.getChannels()) {
817       c.close();
818     }
819   }
820
821   public void disconnectAll() throws IOException{
822     LOG.info("Disconnecting all clients");
823     for(SocketChannel c : server.getChannels()) {
824       c.close();
825     }
826   }
827
828   public void connectAll() throws IOException{
829     LOG.info("Connecting all clients");
830     for(SocketChannel c : server.getChannels()) {
831       c.connect();
832     }
833   }
834
835   public void shutdownAll() throws IOException{
836     LOG.info("Shutting down all clients");
837     for(SocketChannel c : server.getChannels()) {
838       c.close();
839     }
840   }
841
842   public void startAll() throws IOException{
843     LOG.info("Starting all clients");
844     for(SocketChannel c : server.getChannels()) {
845       c.connect();
846     }
847   }
848
849   public void stopAll() throws IOException{
850     LOG.info("Stopping all clients");
851     for(SocketChannel c : server.getChannels()) {
852       c.close();
853     }
854   }
855
856   public void disconnectAll() throws IOException{
857     LOG.info("Disconnecting all clients");
858     for(SocketChannel c : server.getChannels()) {
859       c.close();
860     }
861   }
862
863   public void connectAll() throws IOException{
864     LOG.info("Connecting all clients");
865     for(SocketChannel c : server.getChannels()) {
866       c.connect();
867     }
868   }
869
870   public void shutdownAll() throws IOException{
871     LOG.info("Shutting down all clients");
872     for(SocketChannel c : server.getChannels()) {
873       c.close();
874     }
875   }
876
877   public void startAll() throws IOException{
878     LOG.info("Starting all clients");
879     for(SocketChannel c : server.getChannels()) {
880       c.connect();
881     }
882   }
883
884   public void stopAll() throws IOException{
885     LOG.info("Stopping all clients");
886     for(SocketChannel c : server.getChannels()) {
887       c.close();
888     }
889   }
890
891   public void disconnectAll() throws IOException{
892     LOG.info("Disconnecting all clients");
893     for(SocketChannel c : server.getChannels()) {
894       c.close();
895     }
896   }
897
898   public void connectAll() throws IOException{
899     LOG.info("Connecting all clients");
900     for(SocketChannel c : server.getChannels()) {
901       c.connect();
902     }
903   }
904
905   public void shutdownAll() throws IOException{
906     LOG.info("Shutting down all clients");
907     for(SocketChannel c : server.getChannels()) {
908       c.close();
909     }
910   }
911
912   public void startAll() throws IOException{
913     LOG.info("Starting all clients");
914     for(SocketChannel c : server.getChannels()) {
915       c.connect();
916     }
917   }
918
919   public void stopAll() throws IOException{
920     LOG.info("Stopping all clients");
921     for(SocketChannel c : server.getChannels()) {
922       c.close();
923     }
924   }
925
926   public void disconnectAll() throws IOException{
927     LOG.info("Disconnecting all clients");
928     for(SocketChannel c : server.getChannels()) {
929       c.close();
930     }
931   }
932
933   public void connectAll() throws IOException{
934     LOG.info("Connecting all clients");
935     for(SocketChannel c : server.getChannels()) {
936       c.connect();
937     }
938   }
939
940   public void shutdownAll() throws IOException{
941     LOG.info("Shutting down all clients");
942     for(SocketChannel c : server.getChannels()) {
943       c.close();
944     }
945   }
946
947   public void startAll() throws IOException{
948     LOG.info("Starting all clients");
949     for(SocketChannel c : server.getChannels()) {
950       c.connect();
951     }
952   }
953
954   public void stopAll() throws IOException{
955     LOG.info("Stopping all clients");
956     for(SocketChannel c : server.getChannels()) {
957       c.close();
958     }
959   }
960
961   public void disconnectAll() throws IOException{
962     LOG.info("Disconnecting all clients");
963     for(SocketChannel c : server.getChannels()) {
964       c.close();
965     }
966   }
967
968   public void connectAll() throws IOException{
969     LOG.info("Connecting all clients");
970     for(SocketChannel c : server.getChannels()) {
971       c.connect();
972     }
973   }
974
975   public void shutdownAll() throws IOException{
976     LOG.info("Shutting down all clients");
977     for(SocketChannel c : server.getChannels()) {
978       c.close();
979     }
980   }
981
982   public void startAll() throws IOException{
983     LOG.info("Starting all clients");
984     for(SocketChannel c : server.getChannels()) {
985       c.connect();
986     }
987   }
988
989   public void stopAll() throws IOException{
990     LOG.info("Stopping all clients");
991     for(SocketChannel c : server.getChannels()) {
992       c.close();
993     }
994   }
995
996   public void disconnectAll() throws IOException{
997     LOG.info("Disconnecting all clients");
998     for(SocketChannel c : server.getChannels()) {
999       c.close();
1000    }
1001 }

```

```

68         }
69     }
70     somethingChanged = true;
71     devices.put(id, Device.createDevice(params, this));
72 }
73 else{
74     LOG.log("Received_strange_device_spec_without_id", 0);
75 }
76 }

77 //remove devices that are gone
78 Iterator<String> it = devices.keySet().iterator();
79 while(it.hasNext()){
80     String id = it.next();
81     if(!newIds.contains(id)){
82         it.remove();
83     }
84 }
85 }

87 //notify server that i received a new config if something changed
88 if(somethingChanged) server.newConfig();
89 }

91 private void newConsumptions(Document dom) {
92     NodeList devicesNodeList = dom.getElementsByTagName("item");
93     HashMap<Date, Double> newValues = new HashMap<Date, Double>();
94     LOG.log("values_" + devicesNodeList.getLength(), 20);
95     for(int i=0; i<devicesNodeList.getLength(); i++){
96         NodeList children = devicesNodeList.item(i).getChildNodes();
97         Node timestamp = null;
98         Node value = null;
99         for(int j = 0; j<children.getLength(); j++){
100             if(children.item(j).getnodeName() == "value"){
101                 value = children.item(j);
102             } else if(children.item(j).getnodeName() == "timestamp"){
103                 timestamp = children.item(j);
104             }
105         }
106         if(timestamp == null || value == null){
107             continue;
108         }
109         String time = timestamp.getTextContent();
110         try {
111             Date date = SingletonUtil.instance().gmtDateFormat.parse(time);
112             double v = Double.parseDouble(value.getTextContent());
113             newValues.put(date, v);
114             LOG.log(SingletonUtil.instance().formatTime(date) + "-" + v, 20);
115         } catch (ParseException e) {
116             continue;
117         }
118     }
119     this.server.newConsumptions(newValues, this);
120 }
121 }

123 public void send(String data){
124     server.send(channel, data.getBytes());
125 }
126 }

127 private void removeCommands(){
128     int first = stringBuilder.indexOf("<");
129     for(int i = 0; i < first; i++){
130         if( Character.isWhitespace(stringBuilder.charAt(0)) ){
131             stringBuilder.deleteCharAt(0);
132         } else{
133             System.err.println("Received_data_not_expected");
134             server.closeConnection(channel);
135             return;
136         }
137     }
138 }

139 int last = stringBuilder.indexOf(">");
140 String xml = null;
141 if(last != -1){
142     String root = stringBuilder.substring(1, last).trim();
143     if(!root.endsWith("/")){
144         last = stringBuilder.indexOf(root, last);
145         if(last != -1){
146             last = stringBuilder.indexOf(">", last);
147         }
148     }
149 }
150 }
```

```

152     if(last != -1){
153         xml = stringBuilder.substring(0, last+1);
154         stringBuilder.delete(0, last+1);
155     }
156 }
157 if(xml != null){
158     StringReader reader = new StringReader( xml );
159     InputSource inputSource = new InputSource( reader );
160     Document dom = null;
161     try {
162         dom = domBuilder.parse( inputSource );
163     } catch (Exception e) {
164         e.printStackTrace();
165         server.closeConnection(channel);
166         return;
167     }
168     String root = dom.getDocumentElement().getnodeName();
169     if(root.equals("config")){
170         newConfig(dom);
171     }
172     else if(root.equals("consumptions")){
173         newConsumptions(dom);
174     }
175     else if(root.equals("time")){
176         send("<time>" + SingletonUtil.instance().formatTime(new Date()) + "</time>\n")
177         ;
178     }
179     else{
180         LOG.log("Received_unknown_xml_data", 0);
181         LOG.log(xml, 1);
182     }
183     reader.close();
184 }
185 }
186
187 public HashSet<ShortOffDevice> getShortOffDevices(){
188     HashSet<ShortOffDevice> result = new HashSet<ShortOffDevice>();
189     synchronized(devices){
190         Iterator<Device> it = devices.values().iterator();
191         while(it.hasNext()){
192             Device device = it.next();
193             if(device instanceof ShortOffDevice){
194                 ShortOffDevice d = (ShortOffDevice)device;
195                 result.add(d);
196             }
197         }
198     }
199     return result;
200 }
201
202 public TreeSet<DelayOnDevice> getDelayOnDevices(){
203     TreeSet<DelayOnDevice> result = new TreeSet<DelayOnDevice>(SingletonUtil.instance()
204         () .delayOnComparator);
205     synchronized(devices){
206         Iterator<Device> it = devices.values().iterator();
207         while(it.hasNext()){
208             Device device = it.next();
209             if(device instanceof DelayOnDevice){
210                 DelayOnDevice d = (DelayOnDevice)device;
211                 result.add(d);
212             }
213         }
214     }
215     return result;
216 }
217
218 public String getId() {
219     return this.channel.socket().getRemoteSocketAddress().toString().substring(1);
220 }

```

Listing B.5: Client.java

```

1 package ch.ethz.baumachr.energyProvider;
2
3 import java.io.IOException;
4 import java.util.Calendar;
5 import java.util.Date;
6 import java.util.GregorianCalendar;
7 import java.util.HashMap;
8 import java.util.Observable;

```

```

9  import java.util.Observer;
10 import javax.xml.parsers.DocumentBuilder;
11 import javax.xml.parsers.DocumentBuilderFactory;
12 import javax.xml.parsers.ParserConfigurationException;
13
14 import org.w3c.dom.Document;
15 import org.w3c.dom.Node;
16 import org.w3c.dom.NodeList;
17 import org.xml.sax.SAXException;
18
19 public class Controller implements Runnable, Observer{
20     public static final int LOGLEVEL = 10;
21
22     private final DSServer server;
23     private final IntradayMarketDataPoint[] data;
24     private final HashMap<Date, Double> dataMapping;
25
26     private DocumentBuilder domBuilder;
27
28     private final Logger LOG;
29
30     private boolean doScheduling = false;
31
32     public Controller(DSServer s) {
33         LOG = SingletonUtil.instance().LOG;
34         this.server = s;
35         data = new IntradayMarketDataPoint[48];
36         dataMapping = new HashMap<Date, Double>();
37
38         try {
39             DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
40             dbf.setNamespaceAware(false);
41             dbf.setValidating(false);
42             dbf.setFeature("http://xml.org/sax/features/namespaces", false);
43             dbf.setFeature("http://xml.org/sax/features/validation", false);
44             dbf.setFeature("http://apache.org/xml/features/nonvalidating/load-dtd-grammar",
45                           false);
46             dbf.setFeature("http://apache.org/xml/features/nonvalidating/load-external-dtd",
47                           false);
47             domBuilder = dbf.newDocumentBuilder();
48         } catch (ParserConfigurationException e) {
49             e.printStackTrace();
50         }
51
52         //add myself to receive update calls from the server on config change
53         server.addObserver(this);
54     }
55
56     private boolean getMarketData(){
57         boolean nothingChanged = true;
58         try {
59             //TODO change in data layout
60             if(false){
61                 Document dom = domBuilder.parse("http://www.epexspot.com/en/market-data/intraday
62                 /intraday-table/-/DE");
63                 NodeList rows = dom.getElementsByTagName("tr");
64                 System.out.println(rows.getLength());
65                 for(int i = 3; i < rows.getLength() - 2; i++){
66                     NodeList columns = rows.item(i).getChildNodes();
67                     int index = 0;
68                     int columnIndex = i - 3;
69                     if(data[columnIndex] == null){
70                         Calendar cal = new GregorianCalendar();
71                         cal.setTime(new Date());
72                         cal.set(Calendar.HOUR_OF_DAY, columnIndex);
73                         cal.set(Calendar.MINUTE, 30);
74                         cal.set(Calendar.SECOND, 0);
75                         cal.set(Calendar.MILLISECOND, 0);
76
77                         data[columnIndex] = new IntradayMarketDataPoint();
78                         data[columnIndex].setHour(cal.getTime());
79
80                         cal.add(Calendar.DAY_OF_MONTH, 1);
81                         data[columnIndex + 24] = new IntradayMarketDataPoint();
82                         data[columnIndex + 24].setHour(cal.getTime());
83
84                     }
85                     IntradayMarketDataPoint today = data[columnIndex];
86                     IntradayMarketDataPoint tomorrow = data[columnIndex + 24];
87                     for(int j = 0; j < columns.getLength(); j++){
88                         Node cellNode = columns.item(j);
89                         if(cellNode.getNodeType() == Node.ELEMENT_NODE){
90                             String cell = cellNode.getTextContent().trim();
91
92                         }
93                     }
94
95                     IntradayMarketDataPoint column = tomorrow;
96
97                 }
98             }
99         } catch (Exception e) {
100             e.printStackTrace();
101         }
102     }
103
104     @Override
105     public void update(Observable o, Object arg) {
106         if(doScheduling)
107             return;
108
109         doScheduling = true;
110
111         Thread thread = new Thread(() -> {
112             getMarketData();
113         });
114         thread.start();
115
116         doScheduling = false;
117     }
118
119     @Override
120     public void run() {
121         getMarketData();
122     }
123
124 }
```

```

90         switch(index){
91             case 0: //hour (01-02)
92                 break;
93             case 1: //low today
94             case 8: //low tomorrow
95                 break;
96             case 2: //high today
97             case 9: //high tomorrow
98                 break;
99             case 3: //last today
100                column = today;
101                //break through intended
102            case 10: //last tomorrow
103                try{
104                    double last = Double.parseDouble( cell );
105                    if( !column.isSameLast( last ) ){
106                        LOG.log("Last_for_"
107                            + SingletonUtil.instance().formatTime(column.getDate()) +
108                            "_changed_from_" + column.getLast() + "_to_" + cell , 15 );
109                        column.setLast( Double.parseDouble( cell ) );
110                        nothingChanged = false;
111                    }
112                } catch(NumberFormatException e){}
113                break;
114            case 4: //avg
115                break;
116            }
117        }
118    }
119}
120}
121}
122if( !nothingChanged ){
123    LOG.log("Market_data_changed" , 8 );
124    for(int i = 0; i < data.length; i++){
125        if(data[i].available()){
126            dataMapping.put(data[i].getDate() , data[i].getLast());
127        }
128    }
129    server.newMarketData(dataMapping);
130}
131}
132} catch (SAXException e2) {
133    e2.printStackTrace();
134} catch (IOException e2) {
135    e2.printStackTrace();
136}
137return !nothingChanged;
138}
139
140private void scheduleDevices(){
141    /*LOG.log("Doing scheduling" , 6 );
142    TreeSet<DelayOnDevice> devices = server.getDelayOnDevices();
143    Iterator<DelayOnDevice> it = devices.iterator();
144
145    Calendar startTime = Calendar.getInstance();
146    //settle down first
147    startTime.add(Calendar.SECOND, 600 );
148    while(it.hasNext()){
149        DelayOnDevice dev = it.next();
150        Calendar deadline = dev.getDeadline();
151        if(startTime.after(deadline)){
152            dev.turnOn(deadline.getTime());
153        }
154        else{
155            dev.turnOn(startTime.getTime());
156            startTime.add(Calendar.SECOND, dev.getOnTime());
157        }
158    }*/
159}
160
161@Override
162public void run() {
163    Calendar nextMarketUpdate = Calendar.getInstance();
164    while(true){
165        try {
166            boolean schedule = false;
167            synchronized(this){
168                if(doScheduling) {
169                    doScheduling = false;
170                    schedule = true;
171                }
172            }
173            if(schedule) scheduleDevices();

```

```

174     Calendar now = Calendar.getInstance();
175     long dt = nextMarketUpdate.getTimeInMillis() - now.getTimeInMillis();
176     boolean marketChanged = false;
177     if(dt <= 0){
178         LOG.log("Updateing-market-data", 18);
179         int minutes = 5;
180         marketChanged = getMarketData();
181         nextMarketUpdate = now;
182         nextMarketUpdate.add(Calendar.MINUTE, minutes);
183         dt = minutes * 60 * 1000;
184     }
185
186     LOG.log("Waiting_" + dt/1000 + "s", 25);
187     synchronized(this){
188         if(marketChanged){
189             doScheduling = true;
190         }
191         if(!doScheduling){
192             this.wait(dt);
193         }
194     }
195     } catch (InterruptedException e) {
196         e.printStackTrace();
197     }
198 }
199
200     @Override
201     public void update(Observable arg0, Object arg1) {
202         synchronized(this){
203             this.doScheduling = true;
204             this.notify();
205         }
206     }
207 }
208 }
```

Listing B.6: Controller.java

```

1 package ch.ethz.baumachr.energyProvider;
2
3 import java.util.HashMap;
4 import java.util.Iterator;
5 import java.util.Map;
6
7 public abstract class Device {
8     public static final int LOGLEVEL = 1;
9     protected final String id;
10    protected final HashMap<String, String> params;
11    protected final Client client;
12
13    protected final Logger LOG;
14
15
16    public Device(final HashMap<String, String> params, Client c){
17        this.id = params.get("id");
18        this.params = params;
19        this.client = c;
20        this.LOG = SingletonUtil.instance().LOG;
21        print();
22    }
23
24    public static Device createDevice(HashMap<String, String> params, Client c){
25        String type = params.get("type");
26        if(type == null){
27            return null;
28        }
29        try {
30            if(type.equals("on")){
31                return new DelayOnDevice(params, c);
32            }
33            else if(type.equals("off")){
34                return new ShortOffDevice(params, c);
35            }
36        }
37        catch (WrongParameterException e) {
38            e.printStackTrace();
39            return null;
40        }
41        return null;
42    }
43 }
```

```

44     protected void checkParams(String type, String[] keys) throws
45         WrongParameterException{
46     if (!params.containsKey("type")){
47         throw new WrongParameterException();
48     }
49     if (!params.get("type").equals(type)){
50         throw new WrongParameterException();
51     }
52     for(int i=0; i<keys.length; i++){
53         if (!params.containsKey(keys[i])){
54             throw new WrongParameterException();
55         }
56     }
57 }
58 public void print(){
59     LOG.log("Device_" + id, 5);
60     Iterator<Map.Entry<String, String>> it = params.entrySet().iterator();
61     while (it.hasNext()) {
62         Map.Entry<String, String> pairs = it.next();
63         LOG.log("\t" + pairs.getKey() + " ==> " + pairs.getValue(), 5);
64     }
65 }
66
67 public class WrongParameterException extends Exception {
68     private static final long serialVersionUID = 1L;
69 }
70
71     public String getType(){
72         return params.get("type");
73     }
74
75     public boolean equals(HashMap<String, String> otherParams){
76         if(params.size() != otherParams.size()){
77             return false;
78         }
79         Iterator<String> it = params.keySet().iterator();
80         while(it.hasNext()){
81             String p = it.next();
82             if (!otherParams.containsKey(p)){
83                 return false;
84             }
85             else if (!otherParams.get(p).equals(params.get(p))){
86                 return false;
87             }
88         }
89         return true;
90     }
91 }
```

Listing B.7: Device.java

```

1  package ch.ethz.baumachr.energyProvider;
2
3  import java.text.ParseException;
4  import java.util.Calendar;
5  import java.util.Date;
6  import java.util.GregorianCalendar;
7  import java.util.HashMap;
8
9  public class DelayOnDevice extends Device {
10
11     public static final int LOGLEVEL = 10;
12
13     private final Logger LOG = SingletonUtil.instance().LOG;
14
15     private Date scheduledAt = null;
16
17     private final String[] keys = {
18         "/"startDetection",
19         "/interrupt",
20         "onTime",
21         "slotLength",
22         "startTime"
23     };
24
25     public DelayOnDevice(HashMap<String, String> params, Client c) throws
26         WrongParameterException {
27         super(params, c);
28         checkParams("on", keys);
29     }

```

```

30     public Date getStartTime() {
31         try {
32             return SingletonUtil.instance().gmtDateFormat.parse(params.get("startTime"));
33         } catch (ParseException e) {
34             e.printStackTrace();
35         }
36         return null;
37     }
38
39     public Calendar getDeadline() {
40         Calendar start = new GregorianCalendar();
41         start.setTime(getStartTime());
42         start.add(Calendar.SECOND, Integer.parseInt(params.get("slotLength")) - Integer.
43             parseInt(params.get("onTime")));
44         return start;
45     }
46
47     public int getOnTime() {
48         return Integer.parseInt(params.get("onTime"));
49     }
50
51     public Date getScheduledTime() {
52         return scheduledAt;
53     }
54
55     public void turnOn(Date time) {
56         if(scheduledAt == null) {
57             reschedule(time);
58         }
59     }
60
61     public void reschedule(Date time) {
62         String at = (time != null)? "at=" + SingletonUtil.instance().formatTime(time) + "
63             ":"; client.send("<delayon" + at + ">" + id + "</delayon>\n");
64         LOG.log("Turning_on_device" + id, 4);
65         LOG.log("\t@(" + ((time==null)? "now" : SingletonUtil.instance().formatTime(time)), 4);
66         scheduledAt = time;
67     }

```

Listing B.8: DelayOnDevice.java

```

1 package ch.ethz.baumachr.energyProvider;
2
3 import java.util.HashMap;
4
5 public class ShortOffDevice extends Device {
6
7     private final String[] keys = {
8         "offTime",
9         "slotLength"
10    };
11
12    public ShortOffDevice(HashMap<String, String> params, Client c) throws
13        WrongParameterException {
14        super(params, c);
15        checkParams("off", keys);
16    }
17
18    public void turnOff(int seconds) {
19        String s = (seconds > 0)? "seconds=" + seconds + ":" : "";
20        client.send("<shortoff" + s + ">" + id + "</shortoff>\n");
21    }
22
23    public void turnOn() {
24        client.send("<shortoff_seconds='0'>" + id + "</shortoff>\n");
25    }

```

Listing B.9: ShortOffDevice.java

```

1 package ch.ethz.baumachr.energyProvider;
2
3
4 import java.io.IOException;
5 import java.net.InetSocketAddress;
6 import java.net.Socket;
7 import java.nio.ByteBuffer;

```

```

8 import java.nio.channels.SelectionKey;
9 import java.nio.channels.Selector;
10 import java.nio.channels.ServerSocketChannel;
11 import java.nio.channels.SocketChannel;
12 import java.util.ArrayList;
13 import java.util.Date;
14 import java.util.HashMap;
15 import java.util.HashSet;
16 import java.util.Iterator;
17 import java.util.List;
18 import java.util.Observable;
19 import java.util.Set;
20 import java.util.TreeSet;
21
22 import javax.xml.parsers.ParserConfigurationException;
23
24 public class DSServer extends Observable implements Runnable {
25     public static final int LOGLEVEL = 10;
26
27     private final ServerSocketChannel serverSocket;
28     private final Selector selector;
29     private ByteBuffer reusableBuffer;
30
31     private EnergyProvider gui;
32
33     private List<SocketChannel> clients;
34
35     private HashMap<SocketChannel, List<ByteBuffer>> writeRequests;
36
37     private final Logger LOG;
38
39     public DSServer(int port, EnergyProvider gui) {
40         this.gui = gui;
41         LOG = SingletonUtil.instance().LOG;
42         reusableBuffer = ByteBuffer.allocate(256);
43         ServerSocketChannel tempServerSocket = null;
44         Selector tempSelector = null;
45         try {
46             tempServerSocket = ServerSocketChannel.open();
47             tempSelector = Selector.open();
48             tempServerSocket.configureBlocking(false);
49             tempServerSocket.socket().bind(new InetSocketAddress(port));
50             tempServerSocket.register(tempSelector, SelectionKey.OP_ACCEPT);
51         } catch (IOException e) {
52             e.printStackTrace();
53         }
54         this.serverSocket = tempServerSocket;
55         selector = tempSelector;
56
57         clients = new ArrayList<SocketChannel>();
58
59         writeRequests = new HashMap<SocketChannel, List<ByteBuffer>>();
60     }
61
62     public void send(SocketChannel socket, byte[] data) {
63         sendIt(socket, data);
64
65         // Finally, wake up our selecting thread so it can make the required changes
66         this.selector.wakeup();
67     }
68
69     private void sendIt(SocketChannel socket, byte[] data) {
70         synchronized (this.writeRequests) {
71             List<ByteBuffer> queue = this.writeRequests.get(socket);
72             if (queue == null) {
73                 queue = new ArrayList<ByteBuffer>();
74                 this.writeRequests.put(socket, queue);
75             }
76             queue.add(ByteBuffer.wrap(data));
77         }
78     }
79
80     public HashSet<ShortOffDevice> getShortOffDevices() {
81         HashSet<ShortOffDevice> result = new HashSet<ShortOffDevice>();
82         synchronized (clients) {
83             Iterator<SocketChannel> it = clients.iterator();
84             while (it.hasNext()) {
85                 SocketChannel channel = it.next();
86                 Client c = (Client)channel.keyFor(selector).attachment();
87                 HashSet<ShortOffDevice> l = c.getShortOffDevices();
88                 result.addAll(l);
89             }
90         }
91         return result;

```

```

92     }
93
94     public TreeSet<DelayOnDevice> getDelayOnDevices() {
95         TreeSet<DelayOnDevice> result = new TreeSet<DelayOnDevice>(SingletonUtil.instance
96             () .delayOnComparator);
97         synchronized(clients){
98             Iterator<SocketChannel> it = clients.iterator();
99             while(it.hasNext()){
100                 SocketChannel channel = it.next();
101                 Client c = (Client)channel.keyFor(selector).attachment();
102                 TreeSet<DelayOnDevice> l = c.getDelayOnDevices();
103                 result.addAll(l);
104             }
105             return result;
106         }
107
108     @Override
109     public void run() {
110         try {
111             while (!Thread.interrupted()) {
112
113                 // iterate through the writeRequests List and mark clients that want to write
114                 synchronized(this.writeRequests) {
115                     Iterator<SocketChannel> writeRequestKeys = this.writeRequests.keySet().
116                         iterator();
117                     while (writeRequestKeys.hasNext()) {
118                         SocketChannel c = writeRequestKeys.next();
119                         if (!writeRequests.get(c).isEmpty()){
120                             c.keyFor(selector).interestOps(SelectionKey.OP_WRITE);
121                         }
122                     }
123
124                     //TODO solve this in a better way
125                     //this does not work if every 9 seconds another client connects
126                     // to the select call with a timeout
127                     selector.select(10000);
128
129                     Set<SelectionKey> selected = selector.selectedKeys();
130
131                     // check if the timeout of select expired or if there is a action going on
132                     if(selected.isEmpty()){
133                         // send a ping message to all clients
134                         synchronized(clients){
135                             Iterator<SocketChannel> it = clients.iterator();
136                             while(it.hasNext()){
137                                 SocketChannel c = it.next();
138                                 sendIt(c, "<ping/>\n".getBytes());
139                             }
140                         }
141                         continue;
142                     }
143
144                     // analyze all selected keys
145                     Iterator<SelectionKey> itr = selected.iterator();
146                     while (itr.hasNext()){
147                         SelectionKey key = itr.next();
148                         if(key.isAcceptable()){
149                             accept(key);
150                         }
151                         else if(key.isReadable()){
152                             read(key);
153                         }
154                         else if(key.isWritable()){
155                             write(key);
156                         }
157                         else{
158                             System.err.println("strange");
159                         }
160                         // clear the keys from the set since they are already processed
161                         selected.clear();
162                     }
163                 }
164             }
165             catch(Exception ex){
166                 ex.printStackTrace();
167             }
168         }
169
170     private void accept(SelectionKey key) throws IOException,
171         ParserConfigurationException{
172         SocketChannel c = serverSocket.accept();
173         Socket s = c.socket();

```

```

173     LOG.log(s.getInetAddress().getHostAddress() + ":" + s.getPort() + "-connected", 3)
174     ;
175     c.configureBlocking(false);
176     SelectionKey sk = c.register(selector, SelectionKey.OP_READ);
177     Client client = new Client(this, c);
178     // add the client to the local clients List
179     synchronized(clients){
180         clients.add(c);
181     }
182     // attach the client to the key
183     sk.attach(client);
184
185     gui.connected(client.getId());
186 }
187
188 private void read(SelectionKey key) throws ParserConfigurationException{
189     SocketChannel c = (SocketChannel)key.channel();
190     reusableBuffer.clear();
191     int bytesRead = 0;
192     try {
193         bytesRead = c.read(reusableBuffer);
194     } catch (IOException e) {
195         closeConnection(c);
196         return;
197     }
198     if(bytesRead < 0){
199         closeConnection(c);
200         return;
201     }
202     if(bytesRead > 0){
203         Client client = (Client) key.attachment();
204         byte[] substring = new byte[bytesRead];
205         reusableBuffer.flip();
206         reusableBuffer.get(substring);
207         client.newData(substring);
208     }
209 }
210
211 public void closeConnection(SocketChannel c){
212     SelectionKey sk = c.keyFor(selector);
213     Client clientObject = (Client)sk.attachment();
214     gui.disconnected(clientObject.getId());
215     Socket s = c.socket();
216     LOG.log(s.getInetAddress().getHostAddress() + ":" + s.getPort() + "-disconnected",
217             3);
218     synchronized(writeRequests){
219         writeRequests.remove(c);
220     }
221     clients.remove(c);
222     try {
223         c.close();
224     } catch (IOException e) {
225         e.printStackTrace();
226     }
227     return;
228 }
229
230 private void write(SelectionKey key) throws IOException,
231         ParserConfigurationException{
232     SocketChannel c = (SocketChannel)key.channel();
233     synchronized(writeRequests){
234         List<ByteBuffer> list = writeRequests.get(c);
235         while(!list.isEmpty()){
236             ByteBuffer buf = list.get(0);
237             c.write(buf);
238             if(buf.remaining() > 0) {
239                 // socket's buffer fills up
240                 break;
241             }
242             list.remove(0);
243         }
244         if (list.isEmpty()) {
245             // We wrote away all data, so we're no longer interested
246             // in writing on this socket. Switch back to waiting for
247             // data.
248             key.interestOps(SelectionKey.OP_READ);
249         }
250     }
251     public void newConsumptions(HashMap<Date, Double> newValues, Client c) {
252         gui.addConsumptionValues(newValues, c.getId());
253     }

```

```

254     public void newMarketData(HashMap<Date, Double> dataMapping) {
255         gui.newMarketData(dataMapping);
256     }
257
258     public void newConfig() {
259         this.setChanged();
260         this.notifyObservers();
261         this.clearChanged();
262     }
263 }

```

Listing B.10: DSSServer.java

```

1 package ch.ethz.baumachr.energyProvider;
2
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import java.awt.Dimension;
6 import java.awt.DisplayMode;
7 import java.awt.GraphicsDevice;
8 import java.awt.GraphicsEnvironment;
9 import java.awt.MouseInfo;
10 import java.awt.Point;
11 import java.awt.event.ActionEvent;
12 import java.awt.event.ActionListener;
13 import java.util.Calendar;
14 import java.util.Date;
15 import java.util.HashMap;
16 import java.util.HashSet;
17 import java.util.Iterator;
18 import java.util.TreeSet;
19
20 import javax.swing.BorderFactory;
21 import javax.swing.BoxLayout;
22 import javax.swing.JButton;
23 import javax.swing.JFrame;
24 import javax.swing.JPanel;
25
26 import org.jfree.chart.ChartFactory;
27 import org.jfree.chart.ChartPanel;
28 import org.jfree.chart.JFreeChart;
29 import org.jfree.chart.axis.NumberAxis;
30 import org.jfree.chart.axis.ValueAxis;
31 import org.jfree.chart.plot.XYPlot;
32 import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
33 import org.jfree.data.time.Millisecond;
34 import org.jfree.data.time.Second;
35 import org.jfree.data.time.TimeSeries;
36 import org.jfree.data.time.TimeSeriesCollection;
37 import org.jfree.ui.ApplicationFrame;
38
39 public class EnergyProvider extends ApplicationFrame {
40     private static final long serialVersionUID = 1L;
41
42     private final DSSServer dSSserver;
43     private final TimeSeriesCollection consumptionDataset;
44     private final TimeSeries consumptionSum;
45     private final HashMap<String, TimeSeries> consumptionSeries;
46     private final HashMap<String, Long> consumptionAlreadyAddedUntil;
47     private final TimeSeries marketSeries;
48     private final Logger LOG;
49
50     final private ChartPanel consumptionChartPanel;
51
52     public EnergyProvider(String title) {
53         super(title);
54
55         LOG = SingletonUtil.instance().LOG;
56
57         JButton underrun = new JButton("Underrun");
58         underrun.addActionListener(new ActionListener() {
59             @Override
60             public void actionPerformed(ActionEvent e) {
61                 TreeSet<DelayOnDevice> delayOns = dSSserver.getDelayOnDevices();
62                 Iterator<DelayOnDevice> onIt = delayOns.iterator();
63                 while(onIt.hasNext()) {
64                     DelayOnDevice device = onIt.next();
65                     device.reschedule(null);
66                 }
67                 HashSet<ShortOffDevice> shortOffs = dSSserver.getShortOffDevices();
68                 Iterator<ShortOffDevice> offIt = shortOffs.iterator();
69             }
70         });
71     }

```

```

69         while(offIt.hasNext()){
70             ShortOffDevice device = offIt.next();
71             LOG.log("Turning_off_device_" + device.id, 4);
72             device.turnOn();
73         }
74     });
75 );
76
77 JButton overrun = new JButton("Overrun");
78 overrun.addActionListener(new ActionListener() {
79     @Override
80     public void actionPerformed(ActionEvent e) {
81         HashSet<ShortOffDevice> devices = dSSserver.getShortOffDevices();
82         Iterator<ShortOffDevice> it = devices.iterator();
83         while(it.hasNext()){
84             ShortOffDevice device = it.next();
85             LOG.log("Turning_off_device_" + device.id, 4);
86             device.turnOff(0);
87         }
88     }
89 });
90
91 consumptionSeries = new HashMap<String, TimeSeries>();
92 consumptionAlreadyAddedUntil = new HashMap<String, Long>();
93 consumptionSum = new TimeSeries("Sum");
94 consumptionSum.setMaximumItemCount(400);
95 consumptionDataset = new TimeSeriesCollection(consumptionSum);
96 JFreeChart chart = ChartFactory.createTimeSeriesChart("Energy_Consumption", //title
97             "Time", //label of x-axis
98             "Energy_consumption_[W]", //label of y-axis
99             consumptionDataset,
100            true, //legend
101            true, // tooltips
102            false //urls
103 );
104 consumptionChartPanel = new ChartPanel(createTimeSeriesChart(chart, 5 * 60 *
105             1000, false));
106 consumptionChartPanel.setPreferredSize(new Dimension(600, 250));
107
108 /*this.marketSeries = new TimeSeries("Market");
109 TimeSeriesCollection dataset = new TimeSeriesCollection(this.marketSeries);
110 chart = ChartFactory.createTimeSeriesChart("Intraday Market", //title
111             "Time", //label of x-axis
112             "Cost [Euro/kWh]", //label of y-axis
113             dataset,
114             false, //legend
115             true, // tooltips
116             false //urls
117 );
118 ChartPanel marketChartPanel = new ChartPanel(createTimeSeriesChart(chart, 24 * 60
119             * 60 * 1000, null, true));
120 marketChartPanel.setPreferredSize(new Dimension(600, 250));*/
121
122 JPanel pu = new JPanel();
123 pu.add(underrun);
124 JPanel po = new JPanel();
125 po.add(overrun);
126
127 JPanel mainPanel = new JPanel();
128 mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));
129 mainPanel.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
130 //mainPanel.add(marketChartPanel);
131 mainPanel.add(consumptionChartPanel);
132 mainPanel.add(pu);
133 mainPanel.add(po);
134
135
136 this.setResizable(false);
137 this.getContentPane().add(mainPanel, BorderLayout.CENTER);
138 this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
139
140 //adjust window size to the content.
141 this.pack();
142 positionAtCenterOfActiveScreen();
143
144
145
146 //start other threads
147 dSSserver = new DSSServer(50007, this);
148 Thread dSSthread = new Thread(dSSserver);
149 dSSthread.start();

```

```

150     Controller controller = new Controller(dSSserver);
151     Thread controllerThread = new Thread(controller);
152     controllerThread.start();
153 }
154 /**
155 * Creates a sample chart.
156 *
157 * @param dataset the dataset.
158 *
159 * @return A sample chart.
160 */
161 private JFreeChart createTimeSeriesChart(JFreeChart chart, double xRange, boolean
162                                         shapes) {
163     final XYPlot plot = chart.getXYPlot();
164     //plot.setBackgroundPaint(Color.lightGray);
165     plot.setDomainGridlinePaint(Color.white);
166     plot.setRangeGridlinePaint(Color.white);
167
168     final XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer();
169     renderer.setBaseLinesVisible(true);
170     //renderer.setSeriesLinesVisible(0, true);
171     renderer.setBaseShapesVisible(shapes);
172     //renderer.setSeriesShapesVisible(0, shapes);
173     plot.setRenderer(renderer);
174
175     // change the auto tick unit selection to integer units only...
176     final ValueAxis rangeX = plot.getDomainAxis();
177     //rangeX.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
178     //rangeX.setRange(0, 99);
179     rangeX.setAutoRange(true);
180     rangeX.setFixedAutoRange(xRange); // 5 minutes
181
182     final NumberAxis rangeY = (NumberAxis) plot.getRangeAxis();
183     rangeY.setStandardTickUnits(NumberAxis.createStandardTickUnits());
184
185     //rangeY.setAutoRange(true);
186
187     //chart.setBackgroundPaint(Color.white);
188     return chart;
189 }
190
191 private void positionAtCenterOfActiveScreen(){
192     //compute the center of the screen and move the window there
193     Point mouse = MouseInfo.getPointerInfo().getLocation();
194     GraphicsDevice[] gds = GraphicsEnvironment.getLocalGraphicsEnvironment().getScreenDevices();
195
196     //hack to always open the window on my second screen
197     //mouse.y = 1200;
198     //mouse.x = 10;
199
200     int screen = 0;
201     int h = 0;
202     int w = 0;
203     for(int i=0; i<gds.length; i++){
204         DisplayMode m = gds[i].getDisplayMode();
205         int tempW = m.getWidth() + w;
206         int tempH = m.getHeight() + h;
207         if(mouse.x < tempW && mouse.y < tempH){
208             screen = i;
209             break;
210         }
211         if(mouse.x >= tempW){
212             w += m.getWidth();
213         }
214         if(mouse.y >= tempH){
215             h += m.getHeight();
216         }
217     }
218     LOG.log("Screen " + screen + " width " + w + " height " + h, 10);
219
220     Point p = new Point();
221     GraphicsDevice gd = gds[screen];
222     int screenWidth = gd.getDisplayMode().getWidth();
223     int screenHeight = gd.getDisplayMode().getHeight();
224     int height = this.getHeight();
225     int width = this.getWidth();
226     double centreX = screenWidth / 2 + w;
227     double centreY = screenHeight / 2 + h;
228     p.setLocation(centreX-width/2, centreY-height/2);
229     this.setLocation(p);
230 }
231

```

```

232     public void addConsumptionValues(HashMap<Date, Double> newValues, String id) {
233         final TreeSet<Date> keys = new TreeSet<Date>(newValues.keySet());
234         synchronized(consumptionSeries){
235             TimeSeries clientsSeries= null;
236             if (!consumptionSeries.containsKey(id)){
237                 clientsSeries = new TimeSeries(id);
238                 clientsSeries.setMaximumItemCount(400);
239                 consumptionSeries.put(id, clientsSeries);
240                 consumptionDataset.addSeries(clientsSeries);
241             }
242             else{
243                 clientsSeries = consumptionSeries.get(id);
244             }
245             Iterator<Date> it = keys.iterator();
246
247             while(it.hasNext()){
248                 Date date = it.next();
249                 //hack to remove the newest element (because of dss bug #1299)
250                 if(!it.hasNext()){
251                     break;
252                 }
253                 Calendar cal = Calendar.getInstance();
254                 cal.setTime(date);
255                 long seconds = cal.getTimeInMillis() / 1000;
256                 double consumption = newValues.get(date);
257                 Second s = new Second(date);
258                 if(clientsSeries.getValue(s) == null){
259                     clientsSeries.addOrUpdate(new Second(date), consumption);
260                 }
261
262                 long startAt = 0;
263                 if(consumptionAlreadyAddedUntil.containsKey(id)){
264                     startAt = consumptionAlreadyAddedUntil.get(id);
265                 }
266                 if(startAt == 0 || startAt <= seconds){
267                     if(startAt == 0){
268                         startAt = seconds;
269                     }
270                     for(long i = startAt; i <= seconds; i++){
271                         cal.setTimeInMillis(i * 1000);
272                         s = new Second(cal.getTime());
273                         Number v = consumptionSum.getValue(s);
274                         double value = (v == null)?0:v.doubleValue();
275                         consumptionSum.addOrUpdate(s, value + consumption);
276                     }
277                     consumptionAlreadyAddedUntil.put(id, seconds + 1);
278                 }
279             }
280         }
281         consumptionChartPanel.updateUI();
282     }
283
284     public void newMarketData(HashMap<Date, Double> newValues) {
285         synchronized(marketSeries){
286             Iterator<Date> it = newValues.keySet().iterator();
287             while(it.hasNext()){
288                 Date date = it.next();
289                 double cost = newValues.get(date);
290                 marketSeries.addOrUpdate(new Millisecond(date), cost);
291             }
292         }
293     }
294
295     public static void main(String[] args) {
296         final EnergyProvider energyProvider = new EnergyProvider("Smart_Grid_Control");
297         //display the window
298         energyProvider.setVisible(true);
299     }
300
301     public void connected(String id){
302     }
303
304     public void disconnected(String id) {
305         if(consumptionSeries.containsKey(id)){
306             consumptionDataset.removeSeries(consumptionSeries.get(id));
307             consumptionSeries.remove(id);
308             consumptionAlreadyAddedUntil.remove(id);
309         }
310     }
311 }
312 }
```

Listing B.11: EnergyProvider.java

```

1 package ch.ethz.baumachr.energyProvider;
2
3 import java.lang.reflect.Field;
4 import java.text.SimpleDateFormat;
5 import java.util.Date;
6
7 public class Logger {
8
9     public final int LEVEL = 1;
10
11    private final String className;
12
13    public Logger(){
14        className = this.getClass().getCanonicalName();
15    }
16
17    public synchronized void log(final String s, int level){
18        int LOG_LEVEL = LEVEL;
19        String callerName = "";
20        String methodName = "";
21        int lineNumber = 0;
22        try {
23            throw new Exception();
24        } catch(Exception e){
25            StackTraceElement[] trace = e.getStackTrace();
26            for(int i=0; i<trace.length; i++){
27                if(!className.equals(trace[i].getClassName())){
28                    String fullCallerName = trace[i].getClassName();
29                    String[] callerClassNameArray = fullCallerName.split("\\.");
30                    callerName = callerClassNameArray[callerClassNameArray.length - 1];
31                    methodName = trace[i].getMethodName();
32                    lineNumber = trace[i].getLineNumber();
33                }
34            }
35            try {
36                Field f = Class.forName(fullCallerName).getField("LOG_LEVEL");
37                LOG_LEVEL = f.getInt(null);
38            } catch (Exception e1) {}
39
40            break;
41        }
42    }
43
44    if(level < LOG_LEVEL){
45        // [2011-10-27 09:20:36] [poll:441]
46        final SimpleDateFormat sdf = new SimpleDateFormat("[yyyy-MM-dd HH:mm:ss] -");
47        final Date d = new Date();
48        System.out.print(sdf.format(d));
49        System.out.print("[ " + callerName + ":" + methodName + ":" + lineNumber + " ] -");
50        System.out.println(s);
51    }
52 }
53 }
```

Listing B.12: Logger.java

```

1 package ch.ethz.baumachr.energyProvider;
2
3 import java.text.SimpleDateFormat;
4 import java.util.Calendar;
5 import java.util.Comparator;
6 import java.util.Date;
7 import java.util.Locale;
8 import java.util.TimeZone;
9
10 public class SingletonUtil {
11
12     private static SingletonUtil instance = null;
13
14     public static SingletonUtil instance(){
15         if(instance == null){
16             instance = new SingletonUtil();
17         }
18         return instance;
19     }
20
21     public final SimpleDateFormat gmtDateFormat;
22     public Logger LOG;
23     public final Comparator<DelayOnDevice> delayOnComparator;
24
25     private SingletonUtil(){}

```

```

26     gmtDateFormat = new SimpleDateFormat("EEE, dd MMMM yyyy HH:mm:ss z", Locale.ENGLISH)
27     ; //Wed, 12 Oct 2011 07:12:00 GMT
28     gmtDateFormat.setTimeZone(TimeZone.getTimeZone("GMT"));
29     LOG = new Logger();
30
31     delayOnComparator = new Comparator<DelayOnDevice>() {
32         @Override
33         public int compare(DelayOnDevice d1, DelayOnDevice d2) {
34             Calendar c1 = d1.getDeadline();
35             Calendar c2 = d2.getDeadline();
36             if(c1.before(c2)) return -1;
37             if(c1.after(c2)) return 1;
38             return 0;
39         }
40     };
41
42     public String formatTime(Date time){
43         return gmtDateFormat.format(time);
44     }
45 }
```

Listing B.13: SingletonUtil.java

```

1 package ch.ethz.baumachr.energyProvider;
2
3 import java.util.Date;
4
5 public class IntradayMarketDataPoint {
6     private Date date;
7     private double last;
8     private boolean lastSet;
9
10    public IntradayMarketDataPoint(){
11        lastSet = false;
12    }
13
14    public Date getDate(){
15        return date;
16    }
17    public void setHour(Date d) {
18        date = d;
19    }
20
21    public boolean available(){
22        return lastSet;
23    }
24
25    public void setLast(double l){
26        this.last = l;
27        this.lastSet = true;
28    }
29    public double getLast(){
30        return last;
31    }
32    public boolean isSameLast(double l){
33        return l==last;
34    }
35 }
```

Listing B.14: IntradayMarketDataPoint.java

B.3 Simulation

```

1 import random
2
3 class Device(object):
4     def __init__(self):
5         self.on = random.choice([True, False])
6         self.consumption = random.randint(0, 2000)
7
8     def isOn(self, time):
9         r = random.random()
10        if r > 0.9:
11            self.on = not self.on
```

```

12     return self.on
13
14 def shutOff(self, time):
15     pass
16
17 def powerOn(self, time):
18     pass
19
20 def getConsumption(self, time):
21     return self.consumption if self.isOn(time) else 0
22
23
24 class ShortOff(Device):
25     def __init__(self):
26         super(ShortOff, self).__init__()
27         self.on = True
28         self.offTime = random.randint(60, 60 * 60 * 4)
29         self.slotLength = random.randint(self.offTime, 60 * 60 * 24)
30         self.alreadyOffInSlot = 0
31         self.slotStart = 0
32         self.lastOffInSlot = 0
33
34     def isOn(self, time):
35         if not self.on:
36             offSinceLastOff = (time - self.lastOffInSlot)
37             if self.offTime - (self.alreadyOffInSlot + offSinceLastOff) < 0:
38                 self.alreadyOffInSlot += offSinceLastOff
39                 self.on = True
40         return self.on
41
42     def shutOff(self, time):
43         if self.on:
44             slotEnd = self.slotStart + self.slotLength
45             if slotEnd < time:
46                 self.alreadyOffInSlot = 0
47                 self.slotStart = time
48                 self.lastOffInSlot = time
49                 self.on = False
50             elif self.alreadyOffInSlot < self.offTime:
51                 self.lastOffInSlot = time
52                 self.on = False
53
54     def shutOffPossible(self, time):
55         slotEnd = self.slotStart + self.slotLength
56         return self.on and (slotEnd < time or self.alreadyOffInSlot < self.offTime)
57
58     def powerOn(self, time):
59         if not self.on:
60             offSinceLastOff = (time - self.lastOffInSlot)
61             self.on = True
62             self.alreadyOffInSlot += offSinceLastOff
63
64 class DelayOn(Device):
65     def __init__(self):
66         super(DelayOn, self).__init__()
67         self.slotStartInterval = 60 * 60 * 24
68         self.slotStart = random.randint(0, self.slotStartInterval)
69         self.onTime = random.randint(60, 60 * 60 * 4)
70         self.slotLength = random.randint(self.onTime, 60 * 60 * 8)
71
72         self.actualStart = self.slotStart + self.slotLength - self.onTime
73
74     def isOn(self, time):
75         if time < self.slotStart + self.slotLength - self.onTime:
76             return (self.actualStart <= time) and (time < self.actualStart + self.onTime)
77         else:
78             self.actualStart = time
79             self.slotStart += self.slotLength + random.randint(0, self.slotStartInterval)
80         return True
81
82     def powerOn(self, time):
83         if self.powerOnPossible(time):
84             self.actualStart = time
85             self.slotStart += self.slotLength + random.randint(0, self.slotStartInterval)
86
87     def powerOnPossible(self, time):
88         return (self.slotStart <= time) and (time <= self.slotStart + self.slotLength -
89                                         self.onTime)
90
91     def schedulable(self, time):
92         return time > self.slotStart
93
94 """d = DelayOn()
95 print d.slotStart, d.slotLength, d.onTime

```

```

95
96
97     previous = -1
98     for t in xrange(0, 60 * 60 * 24 * 7, 10):
99         temp = d.getConsumption(t)
100        d.powerOn(t)
101        if previous != temp:
102            print t, d.getConsumption(t)
103            previous = temp"""

```

Listing B.15: device.py

```

1  #!/usr/bin/python
2
3  import random
4  from multiprocessing import Process, Queue
5  import sys
6  from device import Device, ShortOff, DelayOn
7
8
9  numberOfDevices = 200000
10
11 factor = 1730
12 timestep = 60
13 points = [
14     ( 0 * 60 * 60, 49669.6409090909 * factor),
15     ( 1 * 60 * 60, 48519.3590909091 * factor),
16     ( 2 * 60 * 60, 47920.1045454546 * factor),
17     ( 3 * 60 * 60, 47784.0590909091 * factor),
18     ( 4 * 60 * 60, 48360.1181818182 * factor),
19     ( 5 * 60 * 60, 50001.2045454546 * factor),
20     ( 6 * 60 * 60, 54132.3454545455 * factor),
21     ( 7 * 60 * 60, 59275.2136363636 * factor),
22     ( 8 * 60 * 60, 61325.8181818182 * factor),
23     ( 9 * 60 * 60, 62002.9818181818 * factor),
24     (10 * 60 * 60, 63282.1454545454 * factor),
25     (11 * 60 * 60, 64865.1090909091 * factor),
26     (12 * 60 * 60, 64989.1181818182 * factor),
27     (13 * 60 * 60, 64551.2727272727 * factor),
28     (14 * 60 * 60, 63373.5590909091 * factor),
29     (15 * 60 * 60, 62300.2000000000 * factor),
30     (16 * 60 * 60, 62997.7681818182 * factor),
31     (17 * 60 * 60, 65050.3954545455 * factor),
32     (18 * 60 * 60, 65487.3136363636 * factor),
33     (19 * 60 * 60, 64111.5318181818 * factor),
34     (20 * 60 * 60, 61880.1090909091 * factor),
35     (21 * 60 * 60, 58534.3909090909 * factor),
36     (22 * 60 * 60, 56449.8636363636 * factor),
37     (23 * 60 * 60, 52865.5363636364 * factor),
38     (24 * 60 * 60, 49669.6409090909 * factor)
39 ]
40
41 def getAvailability(t):
42     dailyTime = t % (24 * 60 * 60)
43     for (time, value) in points:
44         if time > dailyTime:
45             return oldValue + ( (value - oldValue) * (dailyTime - oldTime) / (time - oldTime)
46         )
47     oldValue = value
48     oldTime = time
49
50
51 def main():
52     devices = []
53     shortOff = []
54     nowOff = {}
55     delayOn = []
56     for j in xrange(0, numberOfDevices):
57         rand = random.choice([1, 2, 3])
58         if rand == 1:
59             shortOff.append(ShortOff())
60         elif rand == 2:
61             delayOn.append(DelayOn())
62         elif rand == 3:
63             devices.append(Device())
64
65     sys.stderr.write("Number_of_delayOn: " + str(len(delayOn)) + "\n")
66     sys.stderr.flush()
67
68     constantConsumption = len(shortOff) * 1000 + len(devices) * 500
69     sys.stderr.write("Constant_consumption_" + str(constantConsumption) + "\n")

```

```

70     sys.stderr.flush()
71
72
73
74     avg = -points[0][1]
75     for (time, value) in points:
76         avg += value
77     avg /= len(points)-1
78
79     alreadyScheduled = {}
80
81     for t in xrange(0, 60 * 60 * 24 * 7, timestep):
82         sys.stderr.write(str(t)+"\n")
83         sys.stderr.flush()
84
85
86         available = getAvailability(t)
87         #noise = random.randint(-5 * 10 ** 5, 5 * 10 ** 5);
88         #available += noise
89
90         consumption = 0
91         for d in devices:
92             consumption += d.getConsumption(t)
93         for d in shortOff:
94             consumption += d.getConsumption(t)
95         for d in delayOn:
96             consumption += d.getConsumption(t)
97         print t, available, consumption, available-consumption
98
99
100    powerOnNumber = 0
101    shutOffNumber = 0
102    if consumption > available:
103        delta = consumption - available
104        shutOffNumber = int(delta / 1000)
105    elif consumption < available:
106        delta = available - consumption
107        powerOnNumber = int(delta / 1000)
108
109    #sys.stderr.write(str(shutOffNumber) + " " + str(powerOnNumber) + "\n")
110    #sys.stderr.flush()
111
112
113    i = 0
114    while i < len(shortOff) and shutOffNumber > 0:
115        if shortOff[i].shutOffPossible(t):
116            shortOff[i].shutOff(t)
117            shutOffNumber -= 1
118            if not i in nowOff:
119                nowOff[i] = True
120            i += 1
121    i = 0
122    lenNowOff = len(nowOff)
123    while i < lenNowOff and powerOnNumber > 0:
124        (key, value) = nowOff.popitem()
125        if not shortOff[key].isOn(t):
126            shortOff[key].powerOn(t)
127            powerOnNumber -= 1
128        i += 1
129
130    counter = 0
131    for d in delayOn:
132        if d.schedulable(t):
133            counter += 1
134            deadline = d.slotStart + d.slotLength - d.onTime
135            startAt = deadline
136            largest = -10000000000000000000
137            for time in xrange(t, deadline, timestep):
138                temp = getAvailability(time) - constantConsumption - d.consumption
139                if time in alreadyScheduled:
140                    temp -= alreadyScheduled[time]
141
142                if temp >= 0:
143                    startAt = time
144                    break;
145                elif temp > largest:
146                    l = deadline - t
147                    if time - t > l / 2:
148                        factor = 1 - (time - t) / l
149                    else:
150                        factor = (time - t) / l
151                        #factor = ((deadline - t) - (time - t)) / (deadline - t)
152                        startAt = time - factor * d.onTime
153                        largest = temp

```

```

154     d.powerOn(startAt)
155     for duringOn in xrange(startAt, startAt + d.onTime, timestep):
156         if duringOn in alreadyScheduled:
157             alreadyScheduled[duringOn] += d.consumption
158         else:
159             alreadyScheduled[duringOn] = d.consumption
160
161     sys.stderr.write("Scheduling " + str(counter) + "-devices\n")
162     sys.stderr.flush()
163
164
165 for t in xrange(60 * 60 * 24 * 7, 2 * 60 * 60 * 24 * 7, timestep):
166     available = getAvailability(t)
167     consumption = 0
168     for d in devices:
169         consumption += d.getConsumption(t)
170     for d in shortOff:
171         consumption += d.getConsumption(t)
172     for d in delayOn:
173         consumption += d.getConsumption(t)
174     print t, available, consumption, available - consumption
175
176 nowOff = {}
177 for t in xrange(2 * 60 * 60 * 24 * 7, 3 * 60 * 60 * 24 * 7, timestep):
178     available = getAvailability(t)
179     consumption = 0
180     for d in devices:
181         consumption += d.getConsumption(t)
182     for d in shortOff:
183         consumption += d.getConsumption(t)
184     for d in delayOn:
185         consumption += d.getConsumption(t)
186     delta = available - consumption
187     print t, available, consumption, delta
188
189 powerOnNumber = 0
190 shutOffNumber = 0
191 if consumption > available:
192     delta = consumption - available
193     shutOffNumber = int(delta / 1000)
194 elif consumption < available:
195     delta = available - consumption
196     powerOnNumber = int(delta / 1000)
197
198
199 i = 0
200 while i < len(shortOff) and shutOffNumber > 0:
201     if shortOff[i].shutOffPossible(t):
202         shortOff[i].shutOff(t)
203         shutOffNumber -= 1
204         if not i in nowOff:
205             nowOff[i] = True
206         i += 1
207
208
209 i = 0
210 lenNowOff = len(nowOff)
211 while i < lenNowOff and powerOnNumber > 0:
212     (key, value) = nowOff.popitem()
213     if not shortOff[key].isOn(t):
214         shortOff[key].powerOn(t)
215         powerOnNumber -= 1
216     i += 1
217
218 i = 0
219 while i < len(delayOn) and powerOnNumber > 0:
220     if delayOn[i].powerOnPossible(t):
221         delayOn[i].powerOn(t)
222         powerOnNumber -= 1
223     i += 1
224
225
226
227
228
229
230
231
232
233 main()

```

Listing B.16: simulation.py

```
1 #!/usr/bin/python
2
3 import sys
4
5 f = open(sys.argv[1], "r")
6 integral = 0
7 for line in f:
8     l = line.split()
9     time = l[0]
10    available = float(l[1])
11    consumption = float(l[2])
12    delta = abs(available - consumption)
13    integral += delta * (6.0 / 360.0)
14    print time, available, consumption, integral
```

Listing B.17: integrate.py