ETH
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

aizo digitalSTROM

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

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 ienergie[1] 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 aizo[2]. Aizo is a start up company that is developing and selling the digitalSTROM home automation system[3].

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

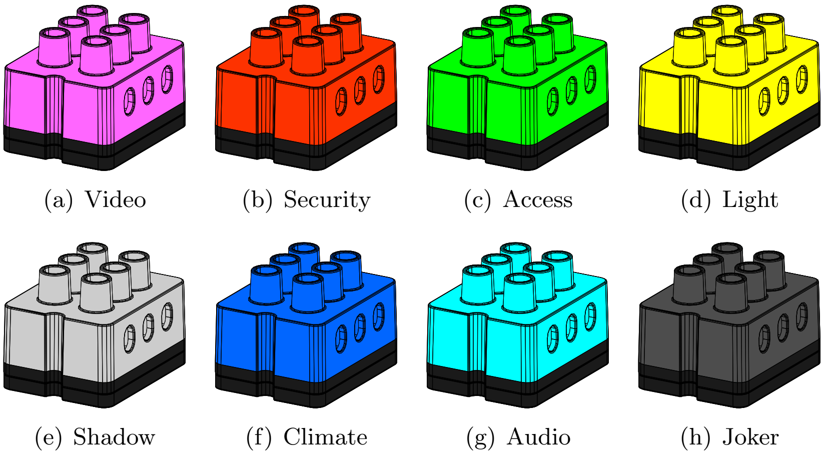
[2] http://www.aizo.com

[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 *dSM*s 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 *dSM*s. 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*


(a) Video (b) Security (c) Access (d) Light

(e) Shadow (f) Climate (g) Audio (h) Joker

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 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

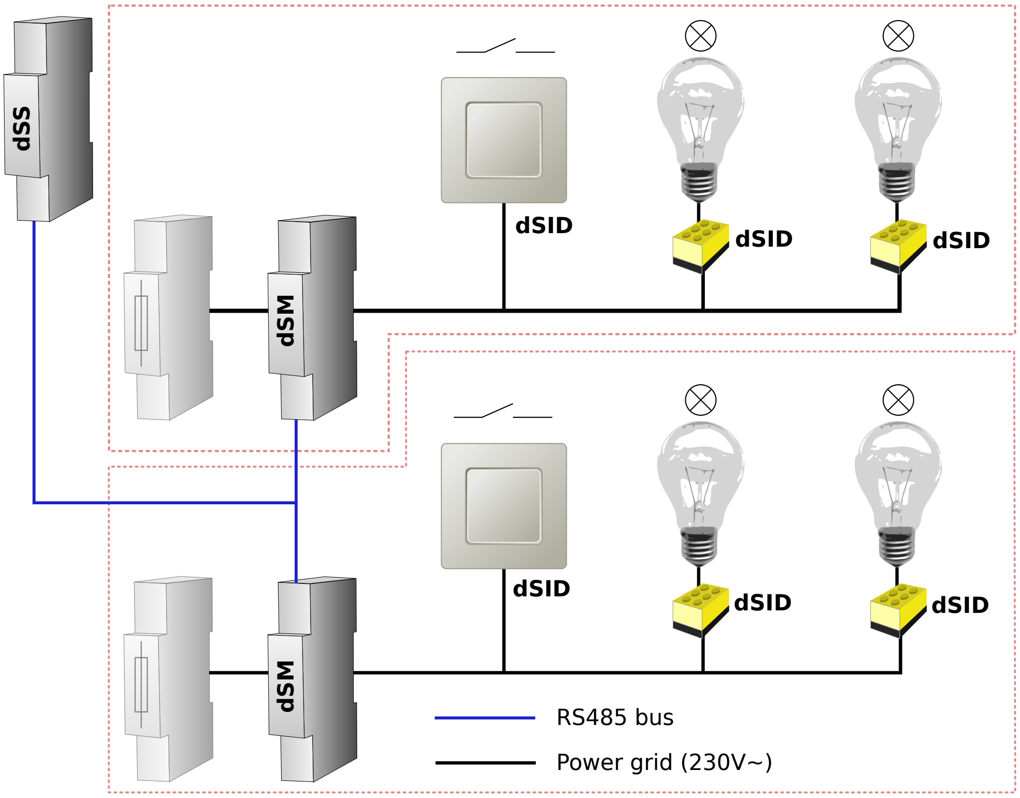
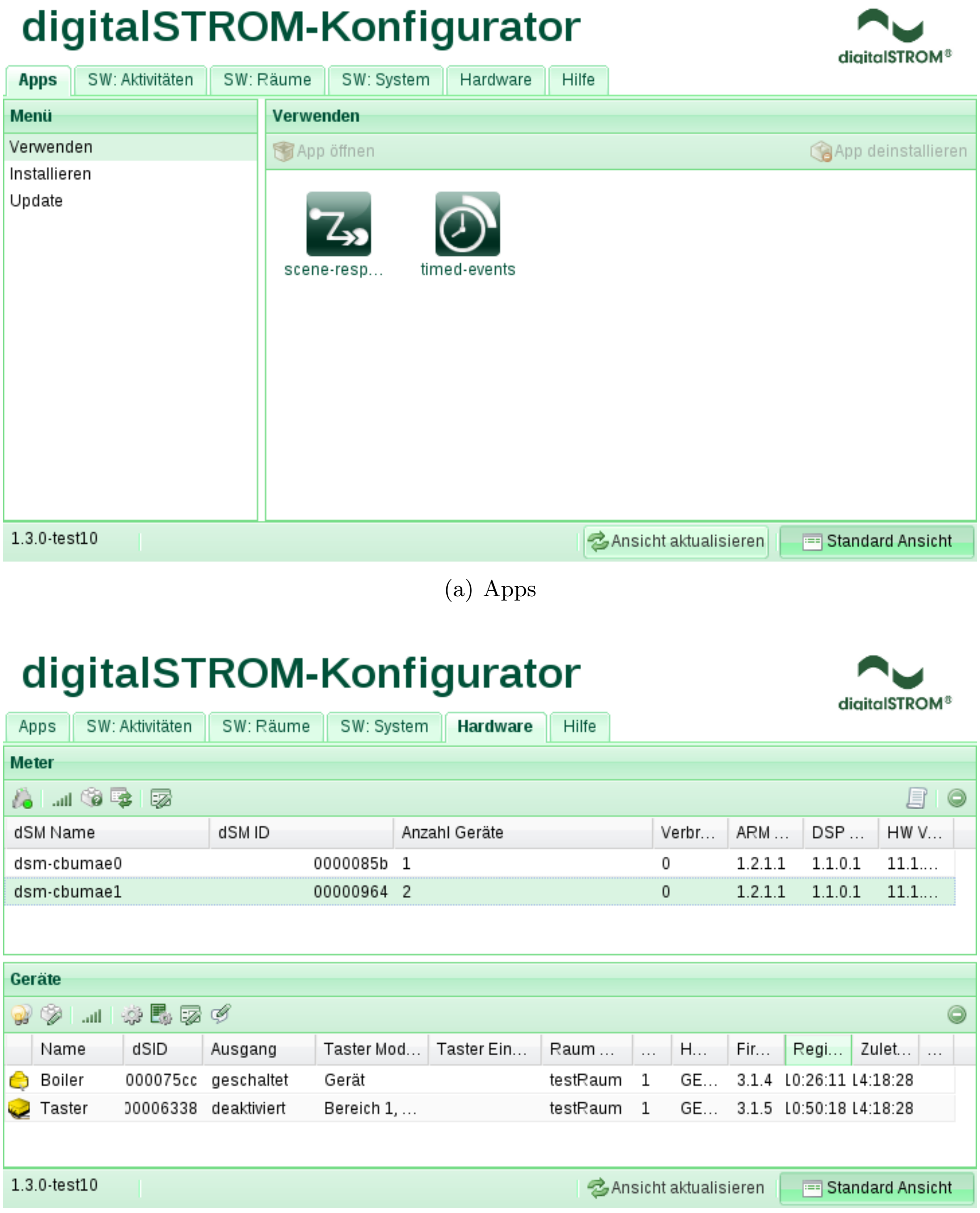


Figure 2.2: Overview of a digitalSTROM installation



(a) Apps

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.…

	Name	dSID	Ausgang	Taster Mod…	Taster Ein…	Raum …	…	H…	Fir…	Regi…	Zulet…	…
	Boiler	000075cc	geschaltet	Gerät		testRaum	1	GE…	3.1.4	10:26:11	14:18:28	
	Taster	00006338	deaktiviert	Bereich 1, …		testRaum	1	GE…	3.1.5	10:50:18	14:18:28	

(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 SpiderMonkey[3] 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.

[1] http://developer.digitalstrom.org/download/dss/1.4/dss-1.4.2-doc/dss-1.4.2-json_api.html
[2] http://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 of[4].

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

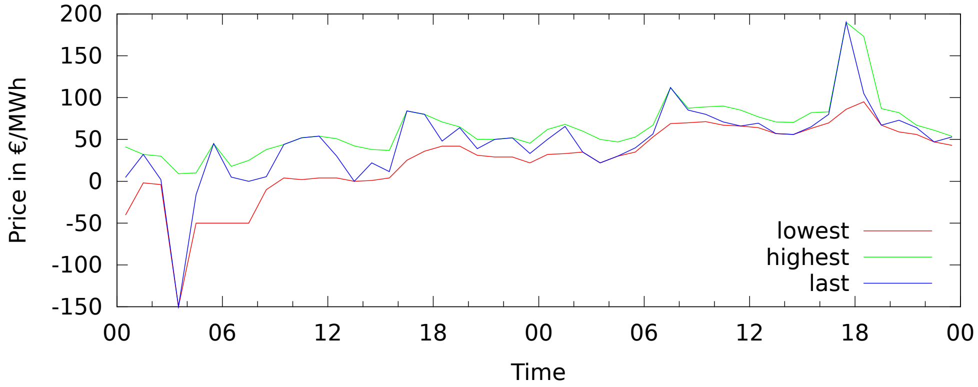


Figure 2.4: Price curves in the time between 27.11.2011 00:00 and 28.11.2011 24:00[5] on the German electricity market. Note that such low negative prices are very rare. Normally the prices stay positive.

[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*.

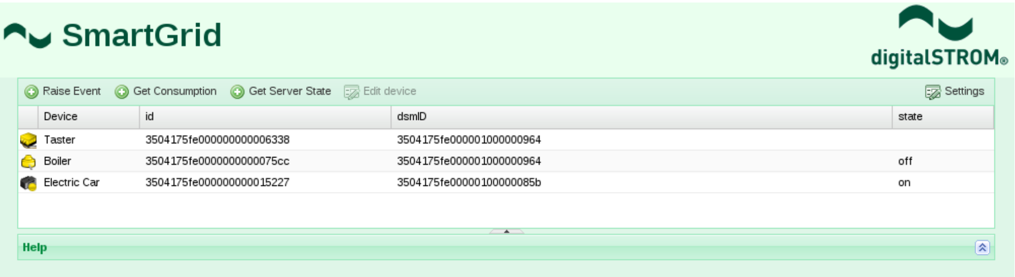
3.1 Idea

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.

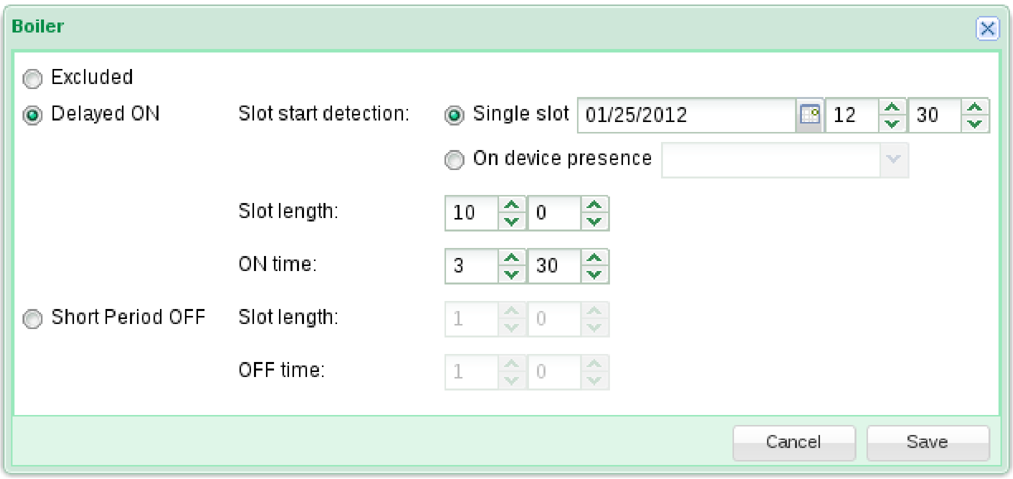
3.2 Configuration

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.



Device	id	dsmID	state
Taster	3504175fe000000000006338	3504175fe000001000000964	
Boiler	3504175fe0000000000075cc	3504175fe000001000000964	off
Electric Car	3504175fe000000000015227	3504175fe00000100000085b	on

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



Boiler

○ Excluded

● Delayed ON — Slot start detection: ● Single slot 01/25/2012 12 30

○ On device presence

Slot length: 10 0

ON time: 3 30

○ Short Period OFF — Slot length: 1 0

OFF time: 1 0

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

Parameter	Description
Type	Delay ON
Slot start	Jan 25, 2012 17:30
Slot length	12 hours.
ON time	4 hours

(b) Freezer

Parameter	Description
Type	Short OFF
Slot length	6 hours
OFF time	20 minutes

Table 3.1: Example configurations of some devices.

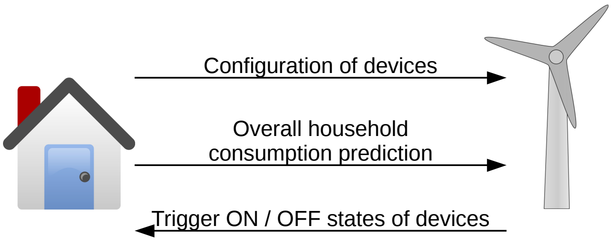

Configuration of devices
Overall household consumption prediction
Trigger ON / OFF states of devices

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 *dSS*. 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 *dSS*'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 *dSS* 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 Sencha[1]. ExtJS4 is a very rich JavaScript library[2] 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.

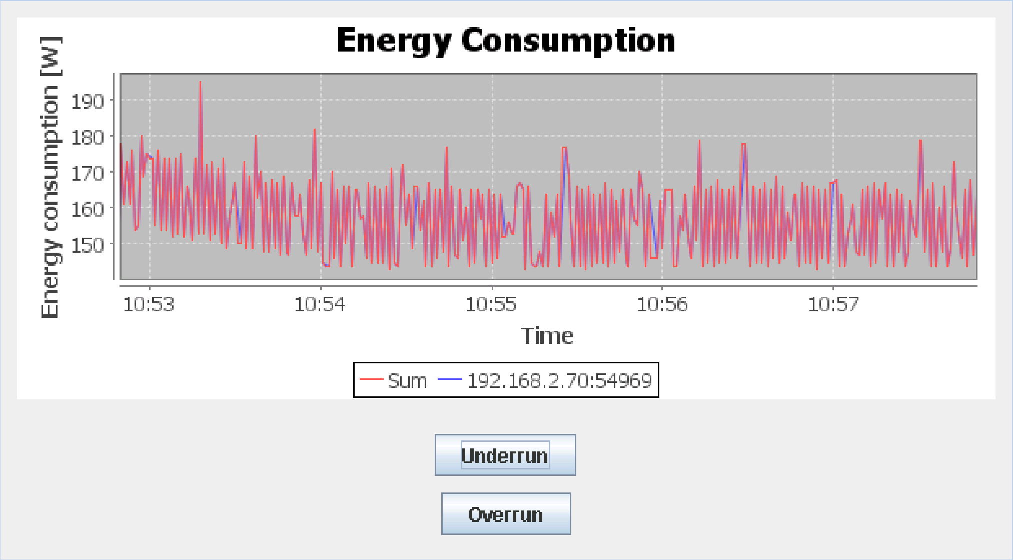


Figure 3.4: Screen shot of the energy provider demo application. Because there is just one client connected the consumption curves for the sum and the single client cannot be differentiated. The library that was used to display the graph is called JFreeChart[3].

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/

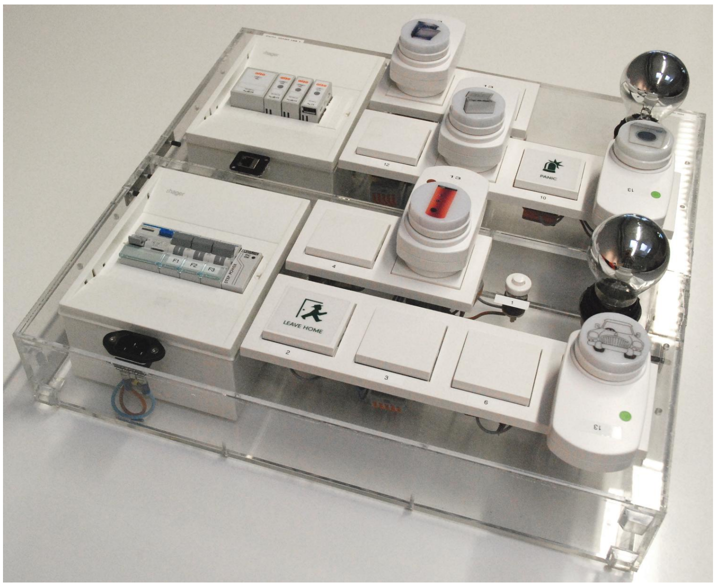


Figure 3.5: Demonstration box containing switches, light bulbs and power sockets. The top left fuse box contains a *dSF*, two *dSMs* and a *dSS*. The bottom left fuse box contains an earth leakage circuit breaker, some fuses and the power supply for the *dSS*.

GT234N freezer[4] 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 roller[5]. 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

Parameter	Description
Type	Delay ON
Slot start	on presence detection of Segway clamp
Slot length	20 hours.
ON time	4 hours

(b) Freezer

Parameter	Value
Type	Short OFF
Slot length	1 hour
OFF time	10 minutes

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.

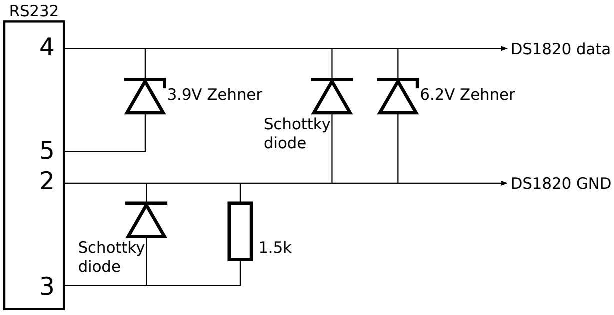


Figure 3.7: Schema to connect the DS1820 temperature sensor to the serial port.

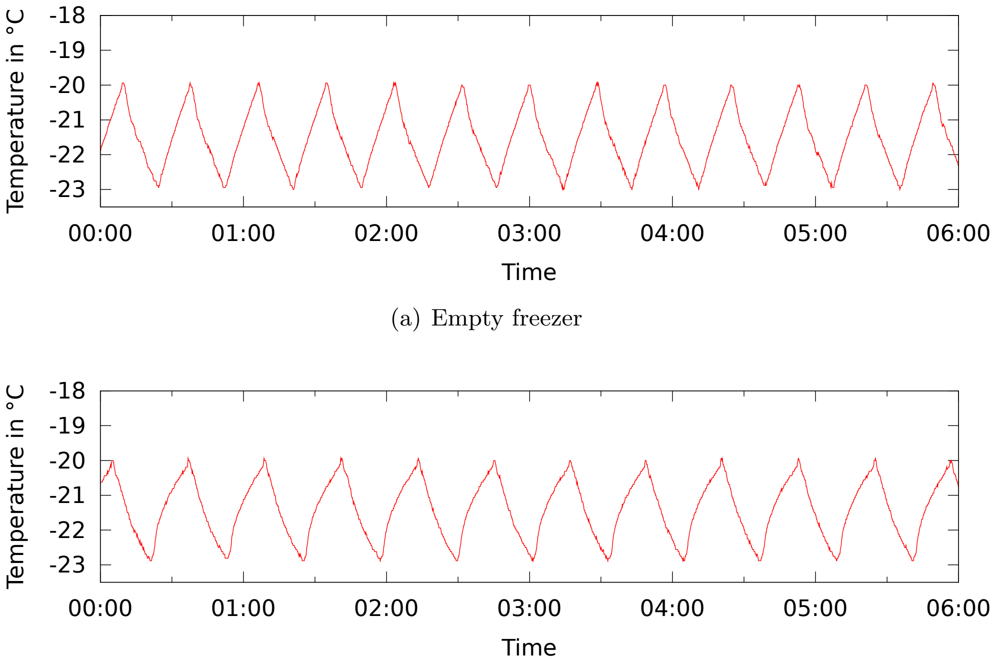
The temperature sensor used was a DS1820 sensor[6] connected to the serial bus of the computer using the schema in the figure 3.7. The open source software to read out the temperature value of the sensor is called digitemp[7].

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/



(a) Empty freezer

(b) Full freezer

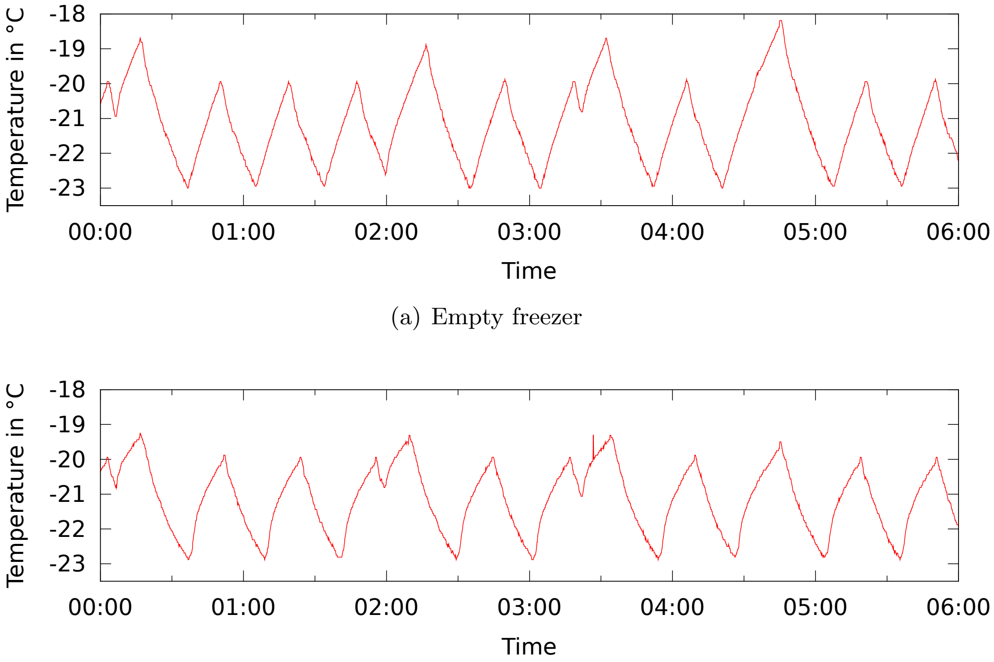
Figure 3.8: Freezer temperatures without load-balancing.

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 weather 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



(a) Empty freezer

(b) Full freezer

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:

$$\int\limits_{t=0}^{\infty} |\text{availability}(t) - \text{consumption}(t)| \, \mathrm{d}t \tag{4.1}$$

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.

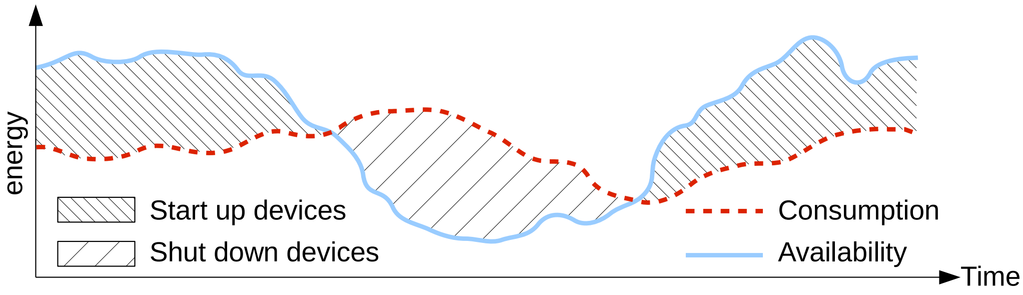


Figure 4.1: Illustration of the idea of the simple algorithm. Just the availabilities and the consumptions are compared.

```
overrun := false;
underrun := false;
measure consumption and availability;
delta :=  abs(consumption - availability);
number := delta / avg_device_consumption;
if consumption > availability then begin
    for i := 0 to number do begin
        if a_device_can_be_turned_off then begin
            turn_off_that_device;
        end;
        else begin
            overrun := true;
            Break;
        end;
    end;
end;
else begin
    for i := 0 to number do begin
        if a_device_can_be_turned_on then begin
            turn_on_device;
        end;
        else begin
            underrun := true;
            Break;
        end;
    end;
end;
if overrun then begin
    turn_off_power_plants;
end;
else if underrun then begin
    turn_on_power_plants;
end;
```

Listing 4.1: Pseudo code of a reacting algorithm.

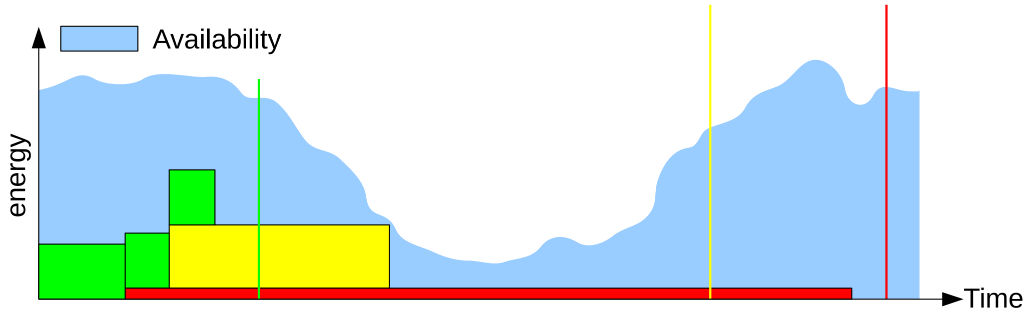


Figure 4.2: The devices have to be scheduled to be finished before their deadlines (vertical lines). The rectangles represent the energy consumption period of a device.

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.com[1] 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:

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

```
run_the_simple_algorithm_ignoring_delay_on_devices;

for i := 0 to length(new_delay_on_device) do begin
    device := new_delay_on_device[i];
    start_at := find_best_starting_point(device);
    device.turn_on(start_at);
end;
```

Listing 4.2: Pseudo code of a scheduling algorithm.

$$\mathrm{f}(t) = \mathrm{prediction\_of\_available}(t) - \mathrm{prediction\_of\_consumption}(t) \tag{4.2}$$

4.2.1 Heuristic to Find the Best Starting Point

The device has to be scheduled in the following interval:

$$I = [\max(\mathrm{now}, \mathrm{earliest\_starting\_time}), \mathrm{latest\_starting\_time}]$$

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

1st case $\mathrm{g}(\max(\mathrm{now}, \mathrm{earliest\_starting\_time}))$ is positive. We are done.

2nd case $\mathrm{g}(t)$ does have one or multiple zero points in the interval I. We can compute one of them assuming $\mathrm{g}(t)$ to be continues 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 $\mathrm{g}(t)$ is not positive in the interval I. In this case we apply another heuristic. We first compute a local maximum of $\mathrm{g}(t)$ 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 $\mathrm{g}(t)$ 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:

$$\begin{aligned} t &= \text{timepoint\_in\_}I\text{\_with\_largest\_availability} \\ e &= \max(\text{now}, \text{earliest\_starting\_time}) \\ l &= \text{latest\_starting\_time} - e \\ f &= \begin{cases} 1 - (t-e)/l & \text{if } t - e > l/2 \\ (t-e)/l & \text{otherwise} \end{cases} \\ \text{start\_at} &= t - \text{f} \cdot \text{onTime} \end{aligned}$$

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.com[2] 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.

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

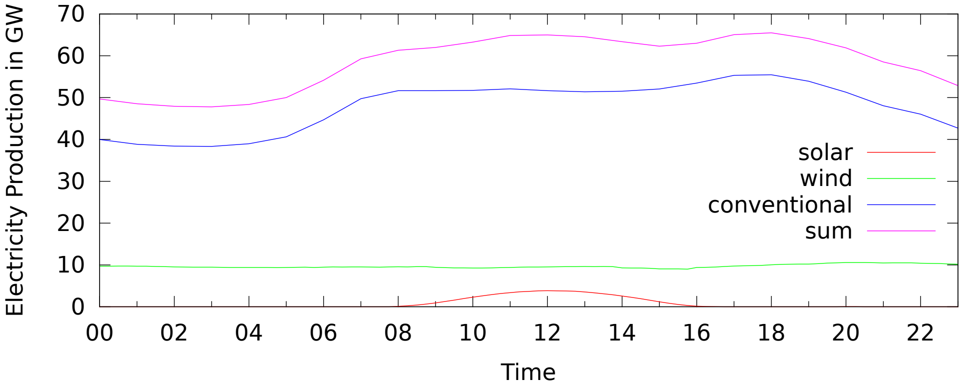


Figure 4.3: Average of the electricity production between November 21, 2011 and December 12, 2011.

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 $^1/_3$ of the devices were excluded from the algorithm, $^1/_3$ were delay ON and the last $^1/_3$ were short OFF devices. There is no particular reason why one should choose $^1/_3$ 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	$[\text{ON time}, 8h]$
Next slot start in	$[0, 24h]$

"short OFF" devices:

Parameter	Range
Consumption	$[0W, 2000W]$.
OFF time	$[1min, 4h]$
Slot length	$[\text{OFF time}, 24h]$

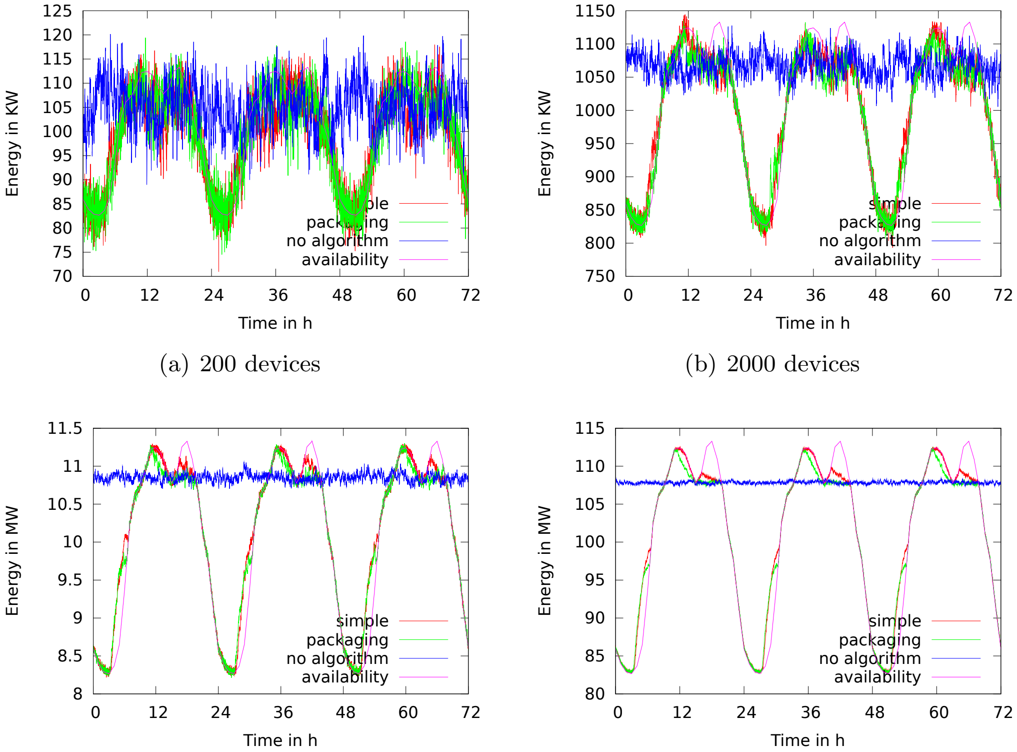
Excluded devices: These devices have a random consumption in the interval $[0W, 2000W]$. They change their ON / OFF state for each evaluation point with the probability $p = 0.9$.

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

$$\int\limits_{t=0}^{x} |\Delta(t)| \, \mathrm{d}t$$

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.



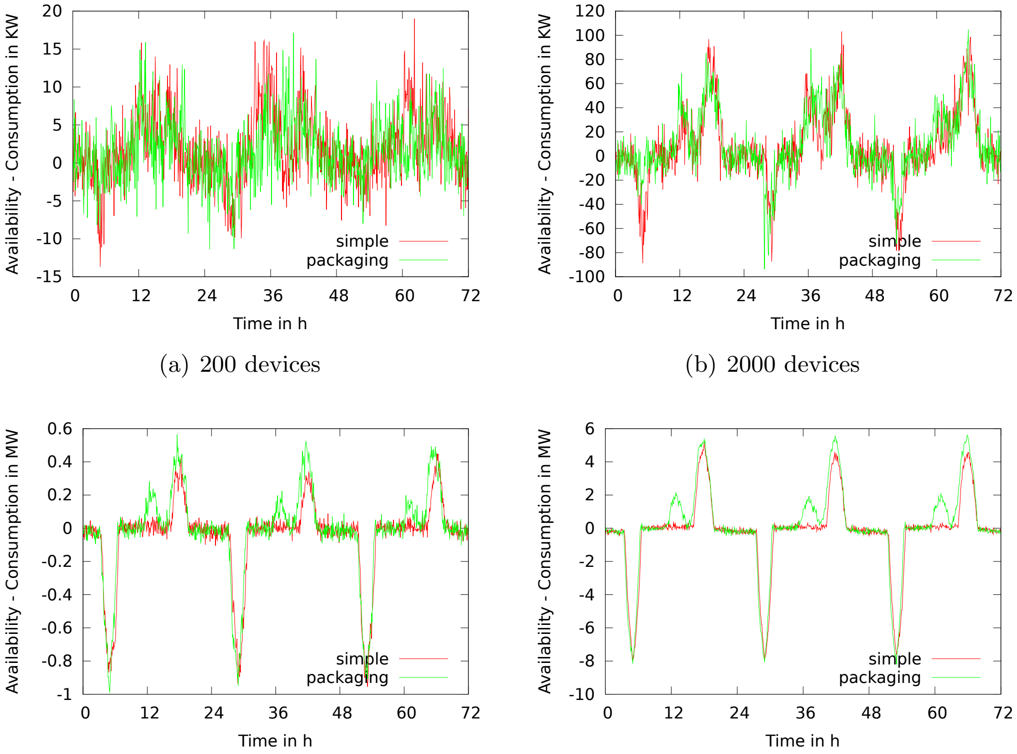
(a) 200 devices

(b) 2000 devices

(c) 20000 devices

(d) 200000 devices

Figure 4.4: Results of simulation for four days



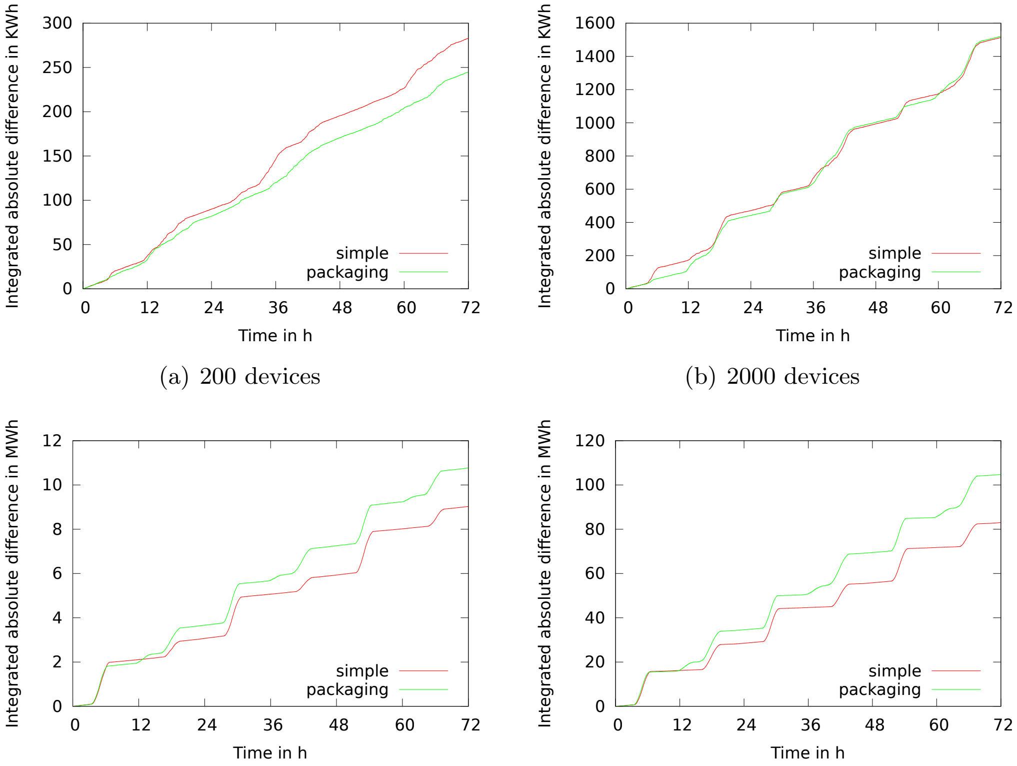
(a) 200 devices

(b) 2000 devices

(c) 20000 devices

(d) 200000 devices

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



(a) 200 devices

(b) 2000 devices

(c) 20000 devices

(d) 200000 devices

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.

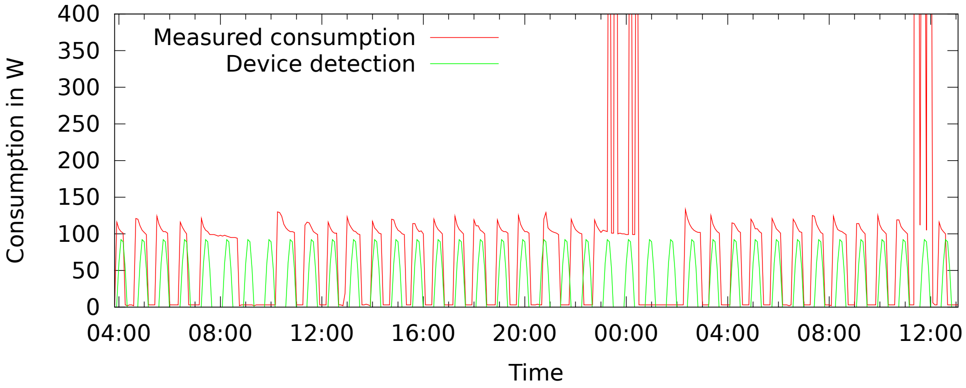
In the current version of digitalSTROM there are power meters deployed directly in the clamps. However there is currently no way to access them. This issue may be resolved in a future version of digitalSTROM. But even if there were accessible power meters at every clamp we could still have some devices 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.

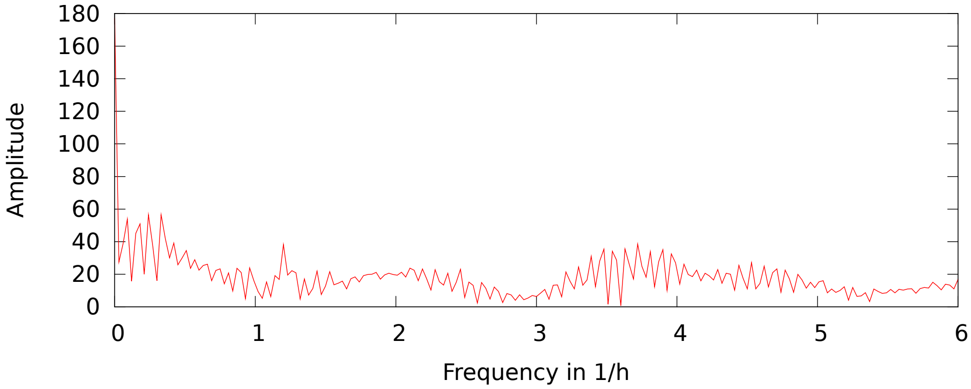
5.1 Detecting Device Consumption Pattern

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 $1.2\frac{1}{\mathrm{h}}$) and transformed back to the time domain.

interesting frequencies are now between $0.5\frac{1}{\mathrm{h}}$ and $2\frac{1}{\mathrm{h}}$ because we expect device patterns in the interval $[0.5\mathrm{h}, 2\mathrm{h}]$. Lower frequencies originate from large peaks and larger frequencies belong to noise. Now we extract the frequency with the largest impact in the interval $[0.5\mathrm{h}, 2\mathrm{h}]$ 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.

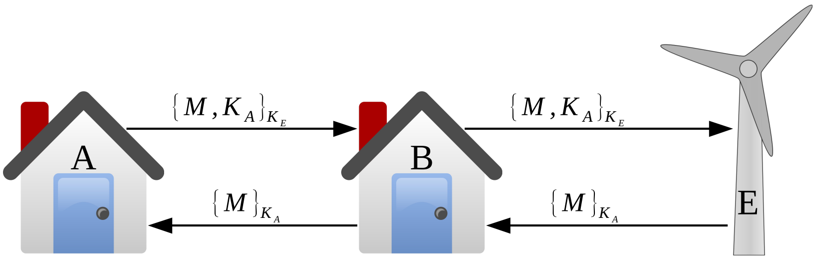


Figure 6.1: Household A generates a random key K_A and adds this key to its message. It decides to send the request to a random node in the crowd. Node B receives the message and decides randomly to send the packet to its destination. It records the route to be able to deliver the response on the same 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 system[1] [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 *UI*. 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 *UI*.
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 *UI* 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 *UI* element to display and edit a time in hours and minutes.
deviceWindow.js	This file describes the main window of the smart-grid configuration *UI*.
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 *UI*.

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 *UI*.
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 do 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 Jython[1] 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 *Interprocess 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 Smart-Grid App

B.1.1 Subscriptions to Events

```
<?xml version="1.0"?>
<subscriptions version="1">
  <subscription event-name="smart-grid" handler-name="javascript">
    <parameter>
      <parameter name="filename1">/usr/share/dss/add-ons/smart-grid/jsonparser.js</
           parameter>
      <parameter name="filename2">/usr/share/dss/add-ons/smart-grid/smartgrid.js</
           parameter>
      <parameter name="filename3">/usr/share/dss/add-ons/smart-grid/rexml.js</
           parameter>
      <parameter name="script_id">smart-grid</parameter>
    </parameter>
  </subscription>
  <subscription event-name="running" handler-name="javascript">
    <parameter>
      <parameter name="filename1">/usr/share/dss/add-ons/smart-grid/jsonparser.js</
           parameter>
      <parameter name="filename2">/usr/share/dss/add-ons/smart-grid/smartgrid.js</
           parameter>
      <parameter name="filename3">/usr/share/dss/add-ons/smart-grid/rexml.js</
           parameter>
      <parameter name="script_id">smart-grid</parameter>
    </parameter>
  </subscription>
</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';
    }
    line = parseInt(callstack[0].split('.js:')[1], 10) + 1;
    LOG.logln('[' + callerName + ':' + line + ']␣' + text);
    /*var i;
    for(i=0; i<callstack.length; i++){
      LOG.logln(callstack[i]);
    }*/
  }
}

/**
 * Checks if the system version matches at least a given version
 * @param version Version to check against (in the format X.Y.Z)
 * @return true if the system version is at least the passed version,
   false otherwise, including the case where the version node doesn't exist
 */
function requireSystemVersion(version) {
  var ndSysVersion = Property.getNode('/system/version/version');
  if (ndSysVersion) {
    var sysVersion, reqVersion, len, i;
    sysVersion = ndSysVersion.getValue().split('.');
    reqVersion = version.split('.');
    len = sysVersion.length < reqVersion.length ? sysVersion.length : reqVersion.length;
    for (i = 0; i < len; ++i) {
      var sys, req;
      sys = parseInt(sysVersion[i], 10);
      req = parseInt(reqVersion[i], 10);
      if (req < sys || (i === len - 1 && req === sys)) {
        return true;
      }
      if (req > sys) {
        return false;
      }
    }
  }
  return false;
} // requireSystemVersion

function timeDriftCheck() {
  var alreadyChecking = Property.getProperty('checkingClockDrift');
  if (alreadyChecking !== null && alreadyChecking) {
    log('Another␣instance␣is␣already␣checking␣the␣clock␣drift␣periodically', 3);
    return;
  }
  Property.setProperty('checkingClockDrift', true);
  timeDriftCheckImpl();
}

function timeDriftCheckImpl() {
  var host = Property.getProperty('serverAddress');
  var port = Property.getProperty('serverPort');
  if (host === null || port === null) {
    log('Server␣not␣specified.', 3);
    return;
  }
  log('Connecting␣to␣' + host + ':' + port, 1);
  var socket = new TcpSocket();
  socket.connect(host, port, function(state) {
    if (state) {
      log('Connected', 1);
      socket.send('<time␣/>', function(bytesSent) {
        log('Sent', 1);
        if (bytesSent > 0) {
          socket.receiveLine(1024, function(data) {
            log('Received␣data', 1);
            log(data, 2);
            var xmlDoc = new REXML(data);
            if (xmlDoc !== null && xmlDoc.rootElement !== null && xmlDoc.rootElement.type === 'element') {
              if (xmlDoc.rootElement.name === 'time') {
                var now = new Date();
                var time = xmlDoc.rootElement.text;
                log('Local␣Time:␣␣' + now.toUTCString(), 0);
                time = new Date(time);
                log('Server␣Time:␣' + time.toUTCString(), 0);
                var drift = Math.abs(now.getTime() - time.getTime());
                log('Drift:␣' + drift + 'ms', 0);
                if (drift < 10000) {
                  Property.setProperty('clockDriftOK', true);
                  startup();
                }
                else {
                  log('Clock␣drift␣is␣too␣large.␣Waiting␣60s.', 0);
                  setTimeout(timeDriftCheckImpl, 60000);
                }
              }
              else {
                log('Root␣element␣not␣time.␣Retrying␣in␣60s.', 0);
                setTimeout(timeDriftCheckImpl, 60000);
              }
            }
            else {
              log('Wrong␣data␣received.␣Retrying␣in␣60s.', 0);
              setTimeout(timeDriftCheckImpl, 60000);
            }
          }, '\n');
        }
        else {
          log('Could␣not␣send␣to␣the␣server.␣Retrying␣in␣60s.', 0);
          setTimeout(timeDriftCheckImpl, 60000);
        }
      });
    }
    else {
      log('Connection␣to␣server␣failed.␣Retrying␣in␣60s.', 0);
      setTimeout(timeDriftCheckImpl, 60000);
    }
  });
}

function getJSONToken() {
  jsonCall('system/login?user=dssadmin&password=dssadmin', function(data) {
    if(data.ok) {
      Property.setProperty('token', data.result.token);
      log('Received␣JSON␣token', 5);
      log('Token:␣' + data.result.token, 15);
    }
    else {
      log('No␣token␣received', 0);
    }
  });
}

function startup() {
  //check if we alread did a proper startup (this function is called again on updateConfig)
  var startupDone = Property.getProperty('startupPerformed');
  if (startupDone === null || !startupDone) {
    var ndVersion = Property.getNode('version');
    if (ndVersion) {
      var version = ndVersion.getValue().split('.');
      if (version.length > 1 && version[0] === '0' && parseInt(version[1], 10) < 9) {
        convertOldEvents();
      }
    }
    var now = new Date();

    //set version number
    Property.setProperty('version', '0.0.1');
    Property.setFlag('version', 'ARCHIVE', true);
    Property.store();

    //check short off devices
    getDevices().perform(function(device) {
      var dsid = device.dsid;
      var type = Property.getProperty('devices/' + dsid + '/config/type');
      log(dsid + '␣' + type, 12);
      if (type !== null && type === 'off') {
        var shortOff = Property.getProperty('devices/' + dsid + '/shortOff');
        if (shortOff !== null && shortOff) {
          log('is␣short␣off', 25);
          var time = Property.getProperty('devices/' + dsid + '/actualShortOffTime');
          var start = Property.getProperty('devices/' + dsid + '/actualShortOffStart');
          if (time === null || start === null) {
            log('Strange:␣shortOff␣is␣true␣but␣the␣values␣are␣not␣set', 0);
          }
          else {
            start = new Date(start);
            var dt = Math.floor((start.getTime() - now.getTime())/1000) + time;
            log('Starting␣off␣device␣' + dsid + '␣in␣' + dt + 's', 3);
            if (dt < 10) {
              var shortOffSlotTime = Property.getProperty('devices/' + dsid + '/shortOffSlotTime');
              Property.setProperty('devices/' + dsid + '/shortOffSlotTime', shortOffSlotTime - dt + 10);
              Property.setFlag('devices/' + dsid + '/shortOffSlotTime', 'ARCHIVE',
                  true);
              Property.store();
              dt = 10; //the system has to settle first
            }
            var event = new TimedEvent('smart-grid', '+' + dt , {action_type: '
                 shortOffReset', dsid: dsid});
            var shortOffEventId = event.raise();

            //not needed on crash
            Property.setProperty('devices/' + dsid + '/shortOffEventId',
                 shortOffEventId);
          }
        }
      }
      else if (type !== null && type === 'on') {
        var on = Property.getProperty('devices/' + dsid + '/delayOn');
        if (on !== null && on) {
          //schedule delayOnReset event
          var onTime = Property.getProperty('devices/' + dsid + '/delayOnTime');
          var minOnTime = Property.getProperty('devices/' + dsid + '/config/onTime');
          onTime = (new Date(onTime)).getTime();
          var offTime = onTime + minOnTime * 1000;
          var deltaT = Math.floor((offTime - now.getTime()) / 1000);
          if (deltaT < 10) {
            deltaT = 10; //the system has to settle first
          }
          log('Stopping␣on␣device␣' + dsid + '␣in␣' + deltaT + 's', 3);
          var stoppEvent = new TimedEvent('smart-grid', '+' + deltaT, {action_type: '
               delayOnReset', dsid: dsid});
          var delayOnResetEventId = stoppEvent.raise();

          Property.setProperty('devices/' + dsid + '/delayOnResetEventId',
               delayOnResetEventId);
        }
        else {
          //schedule delayOnStart event
          var startTime = Property.getProperty('devices/' + dsid + '/config/startTime'
               );
          if (startTime !== null) {
            //schedule delayOnStart event
            var plannedOn = Property.getProperty('devices/' + dsid + '/
                 delayOnPlannedTime');
            plannedOn = new Date(plannedOn);
            var ddt = Math.floor((plannedOn.getTime() - now.getTime()) / 1000);
            if (ddt < 10) {
              ddt = 10; //the system has to settle first
            }
            log('Starting␣on␣device␣' + dsid + '␣in␣' + ddt + 's', 3);
            var startEvent = new TimedEvent('smart-grid', '+' + ddt , {action_type: '
                 delayOnStart', dsid: dsid});
            var delayOnEventId = startEvent.raise();

            //not needed on crash
            Property.setProperty('devices/' + dsid + '/delayOnEventId', delayOnEventId
                 );
          }
        }
      }
    });
    Property.setProperty('startupPerformed', true);
  }

  getJSONToken();

  persistentConnection();


  //start polling
  raiseNextPollEvent();
} // startup

function convertOldEvents() {
  /*TODO*/
}


function jsonCall(path, callback, callbackArgument) {
  var token = '';
  var TOKEN = Property.getProperty('token');
  if (TOKEN !== null) {
    if (path.indexOf('?') === -1) {
      token = '?token=' + TOKEN;
    }
    else {
      token = '&token=' + TOKEN;
    }
  }
  else {
    log('token␣===␣null', 10);
  }
  var data = 'GET␣/json/' + path + token + '␣HTTP/1.0\r\n\r\n';
  log(data, 10);
  httpRequest('localhost', 8088, data, function(line) {
    log(line, 10);
    var data = JSON.parse(line);
    callback(data, callbackArgument);
  });
}

function httpRequest(host, port, data, success, failure) {
  if (failure === null) {
    failure = function() {};
  }
  var socket = new TcpSocket();
  socket.connect(host, port, function(state) {
    if (state) {
      log('Connected', 10);
      socket.send(data, function(bytesSent) {
        if (bytesSent > 0){
          socket.receive(1024, function(data) {
            var lineArray = data.split('\r\n');
            var header = true;
            var i;
            for (i=0; i<lineArray.length; i++) {
              if (!header) {
                log('Data:␣' + lineArray[i], 10);
                success(lineArray[i]);
              }
              else {
                log('Header:␣'+ lineArray[i], 10);
              }
              header = (lineArray[i] !== '');
            }
          });
        }
        else {
          failure();
        }
      });
    }
    else {
      log('Connection␣failed', 0);
      failure();
    }
  });
}

function zeroPad(number) {
  return (number<10) ? '0'+number : number;
}

function poll() {
  var observe = Property.getNode('devicesToObserve');
  if (observe !== null) {
    var i;
    observe = observe.getChildren();
    for (i = 0; i < observe.length; i++) {
      var dsid = observe[i].name;

      var checkIt = false;
      var targets = Property.getNode('devicesToObserve/' + dsid).getChildren();
      var j;
      for (j = 0; j < targets.length; j++) {
        var startTime = Property.getProperty('devices/' + targets[j].name + '/config/
            startTime');
        if (startTime === null) {
          checkIt = true;
        }
      }

      if (checkIt) {
        log('Checking␣device␣' + dsid, 9);
        jsonCall('device/getConfig?class=64&index=0&dsid=' + dsid, checkDeviceCallback
            , dsid);
      }
    }
  }
  function checkDeviceCallback(data, dsid) {
    if (data.ok) {
      log('Device␣' + dsid + '␣is␣present', 5);
      var targets = Property.getNode('devicesToObserve/' + dsid).getChildren();
      var now = new Date();
      var j;
      for (j = 0; j < targets.length; j++) {
        var target = targets[j].name;
        var startTime = Property.getProperty('devices/' + target + '/config/startTime');
        if (startTime === null) {
          Property.setProperty('devices/' + target + '/config/startTime', now.toUTCString());
          Property.setFlag('devices/' + target + '/config/startTime', 'ARCHIVE', true);

          Property.setProperty('devices/' + target + '/deviceDetectedAt', now.toUTCString());
          Property.setFlag('devices/' + target + '/deviceDetectedAt', 'ARCHIVE', true);

          var length = Property.getProperty('devices/' + target + '/config/length');
          var onTime = Property.getProperty('devices/' + target + '/config/onTime');
          var latest = now.getTime() + (length - onTime) * 1000;
          latest = new Date(latest);
          delayOn(target, latest.toUTCString());
        }
      }
      Property.store();
      Property.setProperty('sendRequest', 'newConfig');
    }
    else {
      if (data.message.indexOf('Could␣not␣find␣device␣with␣dsid␣\'') === 0) {
        log('Token␣expired.␣Retrieving␣a␣new␣one', 0);
        getJSONToken();
      }
      else {
        log('Device␣' + dsid + '␣not␣present', 5);
        log('Message:␣' + data.message, 8);
      }
    }
  }




  //check if tcp connection still open
  var time = Property.getProperty('timeLastDataReceived');
  if (time !== null) {
    var now = new Date();
    var dt = Math.round((now.getTime() - time)/1000);
    log('Heard␣nothing␣from␣server␣for␣' + dt + 's', 10);
    var pollInterval = Property.getProperty('pollInterval');
    if (pollInterval === null) {
      pollInterval = 60;
    }
    if (dt > pollInterval * 2) {
      log('Heard␣nothing␣from␣server␣for␣' + dt + 's', 3);
      log('The␣connection␣probably␣died.␣Opening␣a␣new␣one...', 3);
      persistentConnection();
    }
  }
  else{
    log('No␣timeLastDataReceived␣node.␣Opening␣a␣new␣connection...', 3);
    persistentConnection();
  }
} //poll

function raiseNextPollEvent() {
  var pollInterval = Property.getProperty('pollInterval');
  if (pollInterval === null) {
    pollInterval = 60;
  }
  //register new poll event
  var event = new TimedEvent('smart-grid', '+' + pollInterval , {action_type: 'poll'});
  var pollEventId = event.raise();
  Property.setProperty('pollEventId', pollEventId);
}

function updateConfig(config) {
  Property.setProperty('serverAddress', config.server);
  Property.setFlag('serverAddress', 'ARCHIVE', true);
  Property.setProperty('pollInterval', config.poll);
  Property.setFlag('pollInterval', 'ARCHIVE', true);
  Property.setProperty('serverPort', config.port);
  Property.setFlag('serverPort', 'ARCHIVE', true);
  Property.store();

  var pollEventId = Property.getProperty('pollEventId');
  if (pollEventId !== null) {
    Property.getNode('/system/EventInterpreter/ScheduledEvents').removeChild(pollEventId);
  }

  var clockOK = Property.getProperty('clockDriftOK');
  if (clockOK !== null && clockOK) {
    Property.getNode('/scripts/smart-grid').removeChild('clockDriftOK');
    Property.getNode('/scripts/smart-grid').removeChild('checkingClockDrift');
  }
  timeDriftCheck();
}

function shortOff(xmlDoc) {
  var dsid = xmlDoc.rootElement.text;
  var seconds = xmlDoc.rootElement.attribute('seconds');
  var device = getDevices().byDSID(dsid);
  var now = new Date();


  // check that the device is actually configured as shortOff device
  var deviceType = Property.getProperty('devices/' + dsid + '/config/type');
  if (deviceType === null || deviceType !== 'off') {
    log('Device␣' + dsid + '␣is␣not␣configured␣shortOff', 0);
    return true;
  }

  //check that it is not already in shortOff state
  var shortOffProp = Property.getProperty('devices/' + dsid + '/shortOff');
  if (shortOffProp !== null && shortOffProp) {
    if (seconds === '0') {
      log('Turning␣on␣device␣' + dsid, 3);
      device.turnOn();

      var shortOffEventId = Property.getProperty('devices/' + dsid + '/shortOffEventId');
      Property.getNode('/system/EventInterpreter/ScheduledEvents').removeChild(shortOffEventId);

      var lastStart = Property.getProperty('devices/' + dsid + '/actualShortOffStart');
      var time = new Date(lastStart);
      delta = ( now.getTime() - time.getTime() ) / 1000;
      delta = Math.round(delta);

      var lastTime = Property.getProperty('devices/' + dsid + '/actualShortOffTime');

      var shortOffSlotTime = Property.getProperty('devices/' + dsid + '/shortOffSlotTime');
      Property.setProperty('devices/' + dsid + '/shortOffSlotTime', (shortOffSlotTime - lastTime + delta));
      Property.setFlag('devices/' + dsid + '/shortOffSlotTime', 'ARCHIVE', true);

      Property.setProperty('devices/' + dsid + '/shortOff', false);
      Property.setFlag('devices/' + dsid + '/shortOff', 'ARCHIVE', true);

      Property.store();
    }
    else {
      log('Device␣' + dsid + '␣is␣already␣shortOff', 0);
    }
    return true;
  }

  //check that we are allowed to turn off the device under the constraints of the configuration
  var shortOffSlotLength = Property.getProperty('devices/' + dsid + '/config/slotLength');
  var shortOffMaxOff = Property.getProperty('devices/' + dsid + '/config/offTime');
  if (shortOffSlotLength === null || shortOffMaxOff === null) {
    log('Device␣' + dsid + 'is␣not␣configured␣properly', 0);
    return true;
  }

  var shortOffSlotTime = null;
  var shortOffSlotStart = Property.getProperty('devices/' + dsid + '/shortOffSlotStart');
  if (shortOffSlotStart === null) {
    shortOffSlotStart = now;
    shortOffSlotTime = 0;
  }
  else {
    shortOffSlotStart = new Date(shortOffSlotStart);
    var temp = new Date();
    temp.setTime(shortOffSlotStart.getTime() + (shortOffSlotLength * 1000));
    if (temp.getTime() - now.getTime() < 0) {
      shortOffSlotStart = now;
      shortOffSlotTime = 0;
    }
    else {
      shortOffSlotTime = Property.getProperty('devices/' + dsid + '/shortOffSlotTime');
      if (shortOffSlotTime === null) {
        shortOffSlotTime = 0;
      }
    }
  }
  var maxTime = shortOffMaxOff - shortOffSlotTime;
  if (seconds === '') {
    seconds = maxTime;
  }
  else {
    seconds = parseInt(seconds, 10);
  }
  if (seconds > maxTime) {
    log('Time␣too␣large␣for␣device␣' + dsid + '␣max␣' + maxTime, 0);
    return true;
  }

  if (seconds <= 0) {
    log('Time␣' + seconds + '␣not␣turning␣off␣device␣' + dsid, 0);
    return true;
  }

  log('Turning␣off␣device␣' + dsid + '␣for␣' + seconds + 's', 3);
  device.turnOff();

  var event = new TimedEvent('smart-grid', '+' + seconds , {action_type: 'shortOffReset', dsid: dsid});
  var shortOffEventId = event.raise();

  //not needed on crash
  Property.setProperty('devices/' + dsid + '/shortOffEventId', shortOffEventId);

  Property.setProperty('devices/' + dsid + '/shortOffSlotStart', shortOffSlotStart.toUTCString() );
  Property.setFlag('devices/' + dsid + '/shortOffSlotStart', 'ARCHIVE', true);

  Property.setProperty('devices/' + dsid + '/shortOff', true);
  Property.setFlag('devices/' + dsid + '/shortOff', 'ARCHIVE', true);

  Property.setProperty('devices/' + dsid + '/shortOffSlotTime', (shortOffSlotTime + seconds));
  Property.setFlag('devices/' + dsid + '/shortOffSlotTime', 'ARCHIVE', true);

  Property.setProperty('devices/' + dsid + '/actualShortOffTime', seconds);
  Property.setFlag('devices/' + dsid + '/actualShortOffTime', 'ARCHIVE', true);

  Property.setProperty('devices/' + dsid + '/actualShortOffStart', now.toUTCString());
  Property.setFlag('devices/' + dsid + '/actualShortOffStart', 'ARCHIVE', true);

  Property.store();

  return true;
}

function shortOffReset(dsid) {
  var device = getDevices().byDSID(dsid);
  log('Reactivating␣short␣off␣device␣' + dsid, 3);
  device.turnOn();
  Property.setProperty('devices/' + dsid + '/shortOff', false);
  Property.setFlag('devices/' + dsid + '/shortOff', 'ARCHIVE', true);
  Property.store();
}

function delayOn(dsid, atString) {
  var now = new Date();

  //check if the device is configured on
  var type = Property.getProperty('devices/' + dsid + '/config/type');
  if (type === null || type !== 'on') {
    log('Device␣' + dsid + '␣is␣not␣configured␣delayOn', 0);
    return;
  }

  //check if the device is already on
  var delayOnProp = Property.getProperty('devices/' + dsid + '/delayOn');
  if (delayOnProp !== null && delayOnProp) {
    log('Device␣' + dsid + '␣is␣already␣delayOn', 0);
    return;
  }

  var at = now;
  if (atString !== '') {
    var temp = new Date(atString);
    //check that the given time is not in the past
    if (now.getTime() < temp.getTime()) {
      at = temp;
    }
  }

  //check that the current slot is not yet done already
  var slotStartTime = Property.getProperty('devices/' + dsid + '/config/startTime');
  if (slotStartTime === null) {
    log('The␣slot␣is␣already␣done', 0);
    return;
  }
  slotStartTime = new Date(slotStartTime);

  //check if the specified time is not too late
  var slotLength = Property.getProperty('devices/' + dsid + '/config/length');
  var onTime = Property.getProperty('devices/' + dsid + '/config/onTime');
  var latestStart = slotStartTime.getTime() + (slotLength - onTime) * 1000;
  if (at.getTime() > latestStart) {
    latestStart = new Date(latestStart);
    log('Time␣' + at.toUTCString() + '␣is␣too␣late.␣Starting␣at␣' + latestStart.toUTCString(), 0);
    at = latestStart;
  }

  //check that the slot already startet at the time specified
  if (at.getTime() < slotStartTime.getTime()) {
    log('The␣slot␣did␣not␣start␣at␣the␣given␣time', 0);
    return;
  }

  //get event id of the already scheduled event
  var oldEvent = Property.getProperty('devices/' + dsid + '/delayOnEventId');
  if (oldEvent !== null) {
    //delete the event
    Property.getNode('/system/EventInterpreter/ScheduledEvents').removeChild(oldEvent);
    log('Rescheduling␣device␣' + dsid, 3);
  }

  var dt = at.getTime() - now.getTime();
  dt = Math.floor(dt/1000);
  log('Turning␣on␣device␣' + dsid + '␣in␣' + dt + 's', 3);
  if (dt <= 0) {
    delayOnStart(dsid);
  }
  else {
    var event = new TimedEvent('smart-grid', '+' + dt , {action_type: 'delayOnStart', dsid: dsid});
    var delayOnEventId = event.raise();

    //not needed on crash
    Property.setProperty('devices/' + dsid + '/delayOnEventId', delayOnEventId);

    Property.setProperty('devices/' + dsid + '/delayOnPlannedTime', at.toUTCString() );
    Property.setFlag('devices/' + dsid + '/delayOnPlannedTime', 'ARCHIVE', true);
    Property.store();
  }
}

function delayOnStart(dsid) {
  log('Turning␣on␣device␣' + dsid, 3);
  getDevices().byDSID(dsid).turnOn();

  //schedule off event
  var onTime = Property.getProperty('devices/' + dsid + '/config/onTime');
  log('and␣turning␣it␣off␣in␣' + onTime + 's', 3);
  var event = new TimedEvent('smart-grid', '+' + onTime, {action_type: 'delayOnReset', dsid: dsid});
  var delayOnResetEventId = event.raise();

  Property.setProperty('devices/' + dsid + '/delayOnResetEventId', delayOnResetEventId);

  //save everything to be able to recover after crash
  Property.setProperty('devices/' + dsid + '/delayOn', true);
  Property.setFlag('devices/' + dsid + '/delayOn', 'ARCHIVE', true);

  Property.setProperty('devices/' + dsid + '/delayOnTime', (new Date()).toUTCString());
  Property.setFlag('devices/' + dsid + '/delayOnTime', 'ARCHIVE', true);

  Property.setProperty('sendRequest', 'newConfig');
  Property.store();
}

function delayOnReset(dsid) {
  log('Resetting␣delay␣on␣device␣' + dsid, 3);
  getDevices().byDSID(dsid).turnOff();
  Property.setProperty('devices/' + dsid + '/delayOn', false);
  Property.setFlag('devices/' + dsid + '/delayOn', 'ARCHIVE', true);

  Property.getNode('devices/' + dsid + '/config').removeChild('startTime');

  Property.store();
}


function getMeterValues(delta) {
  log('Sending␣meter␣values', 10);
  var now = new Date();
  now = now.getTime();
  var dsms = Apartment.getDSMeters();

  if (delta === null) {
    delta = 10000;
  }
  else {
    delta = Math.ceil(delta / (1000 * 1)); //ms * logging interval
  }
  var result = [];
  var min = delta;

  var firstStamp = null;
  var dataLength = 0;

  if (dsms.length > 0) {
    log('Reading␣meter␣values␣for␣dsm␣' + dsms[0].dsid, 10);
    var dsmData = Metering.getValues(dsms[0].dsid, 'consumption', 1);
    dataLength = dsmData.length;
    if (dataLength > 0) {
      min = Math.min(delta, dataLength);
      firstStamp = dsmData[dataLength - 1].timestamp;
      for (j = dataLength - 1; j >= dataLength - min; j--) {
        var point = dsmData[j];
        var date = new Date(point.timestamp.replace(/-/g, '/'));
        if (date.getTime() <= now) {
          result.push({timestamp: date.toUTCString(), value: point.value});
        }
        else {
          log('Time␣too␣new', 1);
        }
      }
      if (result.length !== min) {
        log("min␣changed", 1);
        min = result.length;
      }

      var i, j;
      for (i = 1; i<dsms.length; i++) {
        log('Reading␣meter␣values␣for␣dsm␣' + dsms[i].dsid, 10);
        dsmData = Metering.getValues(dsms[i].dsid, 'consumption', 1);
        dataLength = dsmData.length;
        j = dataLength;
        while (j > 0 && dsmData[j - 1].timestamp !== firstStamp) {
          j--;
        }
        if ( j === 0 ) {
          log("other␣dsm␣data␣is␣newer", 1);
          continue;
        }
        else if (j !== dataLength) {
          log("newer␣value␣then␣on␣other␣dsm", 10);
          var oldLength = dsmData.length;
          var num = dataLength - j;
          dsmData.splice(j, num);
          dataLength = dsmData.length;
          log('Removing␣' + num + '␣elements␣old:␣'+oldLength+'␣new:␣' + dataLength, 10);
        }

        if (dataLength < min) {
          log('Resizing␣result', 10);
          result.splice(dataLength, min - dataLength);
          min = dataLength;
          log('New␣length␣of␣a␣data␣array:␣' + min, 10);
        }

        for (j = 0; j < min; j++) {
          var point = dsmData[dataLength - 1 - j];
          var date = new Date(point.timestamp.replace(/-/g, '/'));
          if (date.toUTCString() === result[j].timestamp) {
            result[j].value += point.value;
          }
          else {
            log('Timestamp␣does␣not␣match', 10);
          }
        }
      }
    }
  }
  log('Data␣length␣␣␣' + dataLength, 19);
  log('Result␣length␣' + result.length, 19);
  var xml = '<consumptions>' + objToXml(result) + '</consumptions>';
  return xml;
}


function persistentConnection() {
  var socket = new TcpSocket();
  var myLastTime = null;
  var lastMeterTime = null;

  //tried to open a server tcp socket. But just one connection can be accepted and rebuilding the socket always failed.
  //TODO: open bug for this issue.
  /*var control = null;
  function closeServerSocket(){
    if(control !== null){
      //control.close();
      log('WORKS', 0);
    }
    else{
      log('control null', 0);
    }
  }
  var clientS = null;
  function receiveControl(){
    clientS.receiveLine(1024, function(data){
      log('data ' + data, 0);
      //receiveControl();
      clientS.close();
      setTimeout(closeServerSocket, 1000);
    }, '\n');
  }
  function connectionReceived(clientSocket){
    log('Received connection', 0);
    if(socket !== null){
      log('Jupieeeeeeeeeeee', 0);
    }
    //clientSocket.close();
    clientS = clientSocket;
    receiveControl();
    //clientSocket.send('asdfasdf');

    //setTimeout(closeServerSocket, 10000);
    //control.close();
    //control.accept(connectionReceived);
    //buildServerSocket();
    log('Works', 0);
    //control.accept(connectionReceived);
  }
  function buildServerSocket(){
    control = new TcpSocket();
    control.bind(50006, function(state){
      if(state){
        control.accept(connectionReceived);
      }
      else{
```

```
      log('NAK', 0);
    }
  });
}
buildServerSocket();*/

//add a listener to a property to be able to be woken up on a event and send stuff
//this does not work properly either -> going back to polling
/*var listenerId = Property.getProperty('listenerId');
if(listenerId !== null){
  Property.removeListener(listenerId);
}
Property.setProperty('sendRequest', '');
listenerId = Property.setListener('sendRequest', function() {
  var requestType = Property.getProperty('sendRequest');
  if (requestType === 'newConfig') {
    log('Sending new config', 3);
    //the old receive gets somehow killed
    socket.send(getConfigXML(), sent);
  }
  else if (requestType === 'measurements') {
    //log('Sending measurement values', 3);
    socket.send(getMeterValues(), sent);
  }
  else {
    log('Unknown send request ' + requestType, 0);
  }
});
Property.setProperty('listenerId', listenerId);*/

function updateLastReceivedTime() {
  var now = new Date();
  myLastTime = ""+now.getTime();
  Property.setProperty('timeLastDataReceived', myLastTime);
}
function receive(){
  var requestType = Property.getProperty('sendRequest');
  if (requestType === 'newConfig') {
    log('Sending␣new␣config', 3);
    Property.setProperty('sendRequest', '');
    //the old receive gets somehow killed
    socket.send(getConfigXML(), sent);
    return;
  }
  socket.receiveLine(1024, function(data) {
    if (host !== Property.getProperty('serverAddress') || port !== Property.
        getProperty('serverPort')) {
      log('The␣address␣of␣the␣server␣changed.␣Closing␣connection', 3);
      return;
    }

    if (Property.getProperty('timeLastDataReceived') === myLastTime) {
      updateLastReceivedTime();
      if (data === '') {
        //error
        log('Remote␣side␣closed␣connection', 3);
        socket.close();
      }
      else {
        var xmlDoc = new REXML(data);
        if (xmlDoc.rootElement.type === 'element') {
          var name = xmlDoc.rootElement.name;
          if (name === 'ping') {
            var delta = null;
            var now = new Date();
            if (lastMeterTime !== null) {
              delta = now.getTime() - lastMeterTime.getTime();
            }
            else {
              delta = 10000000;
            }
            log(delta + '', 11);
            lastMeterTime = now;
            log('Received␣ping', 20);
            socket.send(getMeterValues(delta), sent);
          }
          else if (name === 'shortoff') {
            shortOff(xmlDoc);
            receive();
          }
          else if (name === 'delayon') {
            delayOn(xmlDoc.rootElement.text, xmlDoc.rootElement.attribute('at'));
            receive();
          }
```

```
            else {
              log(data, 3);
              receive();
            }
          }
          else {
            log(data, 3);
            receive();
          }
        }
      }
      else {
        log('Another␣tcp␣connection␣is␣open.␣I␣will␣close', 3);
      }
    }, '\n');
  }
  function sent(bytesSent) {
    if (bytesSent > 0) {
      log('Sent␣the␣message', 20);
      receive();
    }
    else {
      log('Could␣not␣send␣the␣message', 0);
    }
  }

  var host = Property.getProperty('serverAddress');
  var port = Property.getProperty('serverPort');
  if (host === null || port === null) {
    log('Debug:␣Server␣not␣specified', 3);
    return;
  }
  updateLastReceivedTime();
  socket.connect(host, port, function(state) {
    if (state) {
      log('Connected', 10);
      Property.setProperty('sendRequest', 'newConfig');
      receive();
    }
    else {
      log('Connection␣failed', 0);
    }
  });
} //persistentConnection

function objToXml(obj) {
  var rString='', i;
  if (typeof obj === 'object') {
    if (obj.constructor.toString().indexOf('Array') !== -1) {
      for (i = 0; i < obj.length; i++) {
        rString = rString + ('<item>' + objToXml(obj[i]) + '</item>');
      }
    }
    else {
      for (i in obj) {
        var val = objToXml(obj[i]);
        if (!val) {
          return false;
        }
        rString += '<' + i + '>' + val + '</' + i + '>';
      }
    }
  }
  else if (typeof obj === 'string') {
    rString = obj;
  }
  else if (obj.toString) {
    rString = obj.toString();
  }
  else{
    return false;
  }
  return rString;
}

function getConfigXML() {
  log('Entering␣getConfigXML', 20);
  var devices = [];
  getDevices().perform(function(device) {
    var type = Property.getProperty('devices/' + device.dsid + '/config/type');
    var config = {};
    config.type = type;
    config.id = device.dsid;
    if (type === 'on') {
      config.startTime = Property.getProperty('devices/' + device.dsid + '/config/startTime');
      var delayOn = Property.getProperty('devices/' + device.dsid + '/delayOn');
      if (config.startTime !== null && (delayOn === null || !delayOn) ) {
        config.slotLength = Property.getProperty('devices/' + device.dsid + '/config/length');
        config.onTime = Property.getProperty('devices/' + device.dsid + '/config/onTime');
        devices.push(config);
      }
    }
    else if (type === 'off') {
      config.slotLength = Property.getProperty('devices/' + device.dsid + '/config/slotLength');
      config.offTime = Property.getProperty('devices/' + device.dsid + '/config/offTime');
      devices.push(config);
    }
  });

  var xml = '<config>' + objToXml(devices) + '</config>';
  log('Sending␣config␣to␣the␣server', 10);
  log('Sending␣following␣config␣to␣the␣server␣' + xml, 15);

  return xml;
}

function newConfig(dsid, config) {
  log('New␣config␣for␣' + dsid, 4);

  var configObject = JSON.parse(config);

  var type = configObject.type;
  var oldType = Property.getProperty('devices/' + dsid + '/config/type');

  //reset device
  if (oldType === 'off') {
    var isOff = Property.getProperty('devices/' + dsid + '/shortOff');
    if (isOff !== null && isOff) {
      var resetEventId = Property.getProperty('devices/' + dsid + '/shortOffEventId');
      Property.getNode('/system/EventInterpreter/ScheduledEvents').removeChild(resetEventId);

      //turn the device on again to save the freezer
      shortOffReset(dsid);
    }
  }
  else if (oldType === 'on') {
    //keep the on/off state of the device
    var on = Property.getProperty('devices/' + dsid + '/delayOn');
    if (on !== null && on) {
      var delayOnResetEventId = Property.getProperty('devices/' + dsid + '/delayOnResetEventId');
      Property.getNode('/system/EventInterpreter/ScheduledEvents').removeChild(delayOnResetEventId);
    }
    else {
      var startTime = Property.getProperty('devices/' + dsid + '/config/startTime');
      if (startTime !== null) {
        var delayOnEventId = Property.getProperty('devices/' + dsid + '/delayOnEventId');
        Property.getNode('/system/EventInterpreter/ScheduledEvents').removeChild(delayOnEventId);
      }
    }
    var oldDetectionDevice = Property.getProperty('devices/'  + dsid + '/config/onDetectionDevice');
    if (oldDetectionDevice !== null) {
      oldDetectionDevice = Property.getNode('devicesToObserve/' + oldDetectionDevice);
      if (oldDetectionDevice !== null) {
        oldDetectionDevice.removeChild(dsid);
        if (oldDetectionDevice.getChildren().length === 0) {
          Property.getNode('devicesToObserve').removeChild(oldDetectionDevice);
        }
      }
    }
  }
  var devicesNode = Property.getNode('devices');
  if (devicesNode !== null) {
    devicesNode.removeChild(dsid);
  }

  Property.setProperty('devices/' + dsid + '/config/type', type);
  Property.setFlag('devices/' + dsid + '/config/type', 'ARCHIVE', true);
```

```

    Property.setProperty('devices/' + dsid + '/config/all', config);
    Property.setFlag('devices/' + dsid + '/config/all', 'ARCHIVE', true);

    if (type === 'on') {
      var length = 60 * (configObject.lengthHours * 60 + configObject.lengthMinutes);
      Property.setProperty('devices/' + dsid + '/config/length', length);
      Property.setFlag('devices/' + dsid + '/config/length', 'ARCHIVE', true);

      var onTime = 60 * (configObject.onHours * 60 + configObject.onMinutes);
      Property.setProperty('devices/' + dsid + '/config/onTime', onTime);
      Property.setFlag('devices/' + dsid + '/config/onTime', 'ARCHIVE', true);

      log(configObject.startDetection, 10);
      if (configObject.startDetection === 'singleSlot') {
        var date = configObject.date.replace(/-/g, '/').split('T')[0] + '␣' +
            configObject.startHours + ':' + configObject.startMinutes + ':0';
        var starttime = new Date(date);
        Property.setProperty('devices/' + dsid + '/config/startTime', starttime.
            toUTCString());
        Property.setFlag('devices/' + dsid + '/config/startTime', 'ARCHIVE', true);

        var latest = starttime.getTime() + (length - onTime) * 1000;
        latest = new Date(latest);
        delayOn(dsid, latest.toUTCString());
      }
      else if (configObject.startDetection === 'device') {
        //this is not a new config we just have to detect a slot start
        var device = configObject.onDetectionDevice;
        log('Device␣detection␣with␣device␣' + device, 5);
        Property.setProperty('devicesToObserve/' + device + '/' + dsid, true);
        Property.setFlag('devicesToObserve/' + device + '/' + dsid, 'ARCHIVE', true);

        Property.setProperty('devices/' + dsid + '/config/onDetectionDevice', device);
        Property.setFlag('devices/' + dsid + '/config/onDetectionDevice', 'ARCHIVE',
            true);
      }
    }
    else if (type === 'off') {
      var slotLength = 60 * (configObject.slotLengthHours * 60 + configObject.
          slotLengthMinutes);
      Property.setProperty('devices/' + dsid + '/config/slotLength', slotLength);
      Property.setFlag('devices/' + dsid + '/config/slotLength', 'ARCHIVE', true);

      var offTime = 60 * (configObject.offTimeHours * 60 + configObject.offTimeMinutes);
      Property.setProperty('devices/' + dsid + '/config/offTime', offTime);
      Property.setFlag('devices/' + dsid + '/config/offTime', 'ARCHIVE', true);
    }
    log('New␣config␣saved␣for␣' + dsid, 4);
    Property.setProperty('sendRequest', 'newConfig');

    Property.store();
} //newConfig

function main() {
    if (raisedEvent.name === 'running') {
      // Prepare app
      Property.load();
      LOG.logln('');
      LOG.logln('=======================');
      LOG.logln('#␣␣smart-grid␣running␣␣#');
      LOG.logln('=======================');

      timeDriftCheck();
      return;
    }

    var action_type = raisedEvent.parameter.action_type;
    if (action_type === 'config') {
      log('Debug:␣Updating␣the␣server␣configuration', 5);
      log(raisedEvent.parameter.params, 5);
      updateConfig(JSON.parse(raisedEvent.parameter.params));
      return;
    }


    var clockOK = Property.getProperty('clockDriftOK');
    if (clockOK !== null && clockOK) {
      if (action_type === 'poll') {
        log('Debug:␣Doing␣a␣poll␣to␣the␣electricity␣provider', 20);
        poll();
        raiseNextPollEvent();
      }
      else if (action_type === 'configDevice') {
```

```
      newConfig(raisedEvent.parameter.deviceId, raisedEvent.parameter.config);
    }
    else if (action_type === 'shortOffReset') {
      shortOffReset(raisedEvent.parameter.dsid);
    }
    else if (action_type === 'delayOnStart') {
      delayOnStart(raisedEvent.parameter.dsid);
    }
    else if (action_type === 'delayOnReset') {
      delayOnReset(raisedEvent.parameter.dsid);
    }
    else {
      log('Debug:␣Strange:␣received␣event␣not␣prepared␣for', 0);
      log('Debug:␣smart-grid␣main␣call␣with␣action␣type:␣'
        + raisedEvent.parameter.action_type
        + (raisedEvent.parameter.params === undefined ? ''
          : '␣and␣params:␣'
          + raisedEvent.parameter.params.toString()
        ), 0
      );
    }
  }
  else {
    log('Clock␣not␣verified␣to␣have␣the␣correct␣time.␣Waiting.␣' + action_type, 0);
  }
} // main

main();
```

Listing B.2: scripts/smartgrid.js

B.1.3 User Interface

```
Ext.define('DSS.addon.SmartGrid.ConfigWindow', {
  extend: 'Ext.window.Window',
  title: 'SmartGrid',
  layout: 'fit',
  closeAction: 'hide',

  constructor: function(config){
    this.initConfig(config);
    this.callParent(arguments);
  },

  /** Server Address field */
  serverAddress: null,

  /** Form Panel */
  formPanel: null,

  /** Boolean to remember if the data is already fetched */
  fetchedData: false,


  initComponent: function(){
    var me = this;

    Ext.define('configPanel', {
      extend: 'Ext.form.Panel',
      bodyPadding: 5,  // Don't want content to crunch against the borders
      width: 300,
      items: [
        {
          name: 'serverAddress',
          fieldLabel: _("Server␣address"),
          xtype: 'textfield',
          allowBlank: false
        }, {
          name: 'serverPort',
          fieldLabel: _("Server␣port"),
          xtype: 'numberfield',
          allowBlank: false,
          minValue: 0,
          maxValue: 65535,

          // Remove spinner buttons, and arrow key and mouse wheel listeners
          hideTrigger: true,
          keyNavEnabled: false,
          mouseWheelEnabled: false
        }, {
          name: 'pollInterval',
          fieldLabel: _("Poll␣interval"),
          xtype: 'numberfield',
          minValue: 10, //prevents lower values than 10
          allowBlank: false,

          // Remove spinner buttons, and arrow key and mouse wheel listeners
          hideTrigger: true,
          keyNavEnabled: false,
          mouseWheelEnabled: false
        }
      ],

      /** footer bar */
      fbar: {
        items: [
          {
            text:_("Cancel"),
            id: 'btn-cancel'
          },
          {
            text: _("Save"),
            id: 'btn-save'
          }
        ]
      },

      constructor: function(config){
        this.initConfig(config);
        this.callParent(arguments);
      },

      initComponent: function(){
        var me = this;
        me.addEvents({
          eventhide: true
        });

        me.callParent(arguments);
        me.initPage();
      },


      initPage: function() {
        var me = this;

        Ext.getCmp('btn-save').handler = function() {
          var form = me.getForm();
          if (form.isValid()) {
            me.saveIt(form);
          }
        };
        Ext.getCmp('btn-cancel').handler = function() {
          me.fireEvent('eventhide');
        };
      },

      saveIt: function(form){
        var me = this;

        var data = form.getFieldValues();
        var serverAddress = me.down('[name=serverAddress]');
        var pollInterval = me.down('[name=pollInterval]');
        var serverPort = me.down('[name=serverPort]');
        serverAddress.resetOriginalValue();
        pollInterval.resetOriginalValue();
        serverPort.resetOriginalValue();


        var params = {
          poll: data.pollInterval,
          server: data.serverAddress,
          port: data.serverPort
        };

        var event = Ext.create('DSS.json.Event', {name: 'smart-grid'});
        event.raise({
          action_type: 'config',
          params: Ext.JSON.encode(params)
        }, {
          success: function(){
            me.fireEvent('eventhide');
          },
          failure: function() {
            Ext.Msg.alert(_('Error'), _('Couldn\'t␣create␣timed␣event␣on␣server'));
          }
        });
      },

      getField: function(path, success){
        var me = this;
        Ext.Ajax.request({
          disableCaching: true,
          method: 'GET',
          timeout: 20000,
          url: '/json/' + path,
          success: function(response){
            var data = Ext.JSON.decode(response.responseText);
            if(data.ok){
              success(data.result.value);
            }
            else{
              me.enable();
            }
          },
          failure: function(){
            me.enable();
          }
        });
      },


      beforeShow: function(){
        var me = this;
        if(!me.fetchedData){
          me.disable();
          me.fetchedData = true;
          me.getField("property/getInteger?path=/scripts/smart-grid/pollInterval",
               function(data){
            var pollInterval = me.down('[name=pollInterval]');
            pollInterval.setRawValue(data);
            pollInterval.resetOriginalValue();
            me.getField("property/getString?path=/scripts/smart-grid/serverAddress",
                 function(data){
              var serverAddress = me.down('[name=serverAddress]');
              serverAddress.setRawValue(data);
              serverAddress.resetOriginalValue();
              me.getField("property/getInteger?path=/scripts/smart-grid/serverPort",
                   function(data){
                var serverPort = me.down('[name=serverPort]');
                serverPort.setRawValue(data);
                serverPort.resetOriginalValue();
                me.enable();
              });
            });
          });
        }
        else{
          me.getForm().reset();
        }
      }
    });
    formPanel = Ext.create('configPanel', { property: me.property });
    me.items = formPanel;
    me.items.on({
      eventhide: function(){
        me.hide();
      }
    });

    me.addListener( 'beforeshow', function(){
      formPanel.beforeShow();
    });

    me.callParent(arguments);
  }
});
```

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

```
Ext.define('DSS.addon.SmartGrid.DeviceWindow', {
  extend: 'Ext.window.Window',
  title: 'SmartGrid',
  layout: 'fit',
  closeAction: 'hide',

  /** The store object is received with the config parameter in the constructor */
  store: null,

  /** currently edited device */
  device: null,

  constructor: function(config){
    this.initConfig(config);
    this.callParent(arguments);
  },

  items: [
    {
      xtype: 'form',
      id: 'form',
      bodyPadding: 5,
      items: [
        {
          boxLabel  : _("Excluded"),
          name      : 'type',
          inputValue  : 'exclude',
          id        : 'exclude',
          xtype     : 'radiofield',
          width     : 130
        },
        {
          xtype : 'container',
          layout  : 'column',
          items : [
            {
              boxLabel  : _("Delayed␣ON"),
              name      : 'type',
              inputValue  : 'on',
              id        : 'on',
              xtype     : 'radiofield',
              width     : 130
            },
            {
              xtype:'container',
              id:'onContainer',
              items: [
                {
                  xtype : 'fieldcontainer',
                  fieldLabel  : _("Slot␣start␣detection"),
                  labelWidth: 120,
                  items: [
                    {
                      xtype : 'container',
                      layout  : 'column',
                      items: [
                        {
                          boxLabel: _("Single␣slot"),
                          id      : 'onSingleSlot',
                          name    : 'startDetection',
                          inputValue: 'singleSlot',
                          xtype : 'radiofield'
                        },
                        {
                          xtype: 'container',
                          id: 'startTimeContainer',
                          layout: 'column',
                          items: [
                            {
                              id:'onSingleSlotDate',
                              name: 'date',
                              xtype: 'datefield',
                              margin: '0␣0␣0␣5',
                              allowBlank: false
                            },
                            {
                              id:'onSingleSlotTime',
                              name: 'start',
                              xtype: 'dssTimeSelection'
                            }
                          ]
                        }
                      ]
                    },
                    {
                      xtype: 'container',
                      layout: 'column',
                      id: 'startDetectionContainer',
              items: [
                {
                  boxLabel: _("On␣device␣presence"),
                  id    : 'onDetection',
                  name  : 'startDetection',
                  inputValue: 'device',
                  xtype : 'radiofield',
                  margin  : '0␣5␣0␣0'
                }
              ]
            }
          ]
        },
        {
          xtype: 'fieldcontainer',
          labelWidth: 120,
          fieldLabel: _("Slot␣length"),
          items: [
            {
              id: 'onSlotLength',
              name: 'length',
              xtype: 'dssTimeSelection',
              type: 'length'
            }
          ]
        },
        {
          xtype: 'fieldcontainer',
          labelWidth: 120,
          fieldLabel: _("ON␣time"),
          items: [
            {
              id: 'onOnTime',
              name: 'on',
              xtype: 'dssTimeSelection',
              type: 'length'
            }
          ]
        }/*,
        {
          xtype : 'fieldcontainer',
          fieldLabel  : _("Interruption allowed"),
          labelWidth: 120,
          items: [
            {
              boxLabel: _("Yes"),
              id    : 'interruptTrue',
              name  : 'interrupt',
              inputValue: "yes",
              xtype : 'radiofield',
            },
            {
              boxLabel: _("No, device needs to run in a row"),
              id    : 'interruptFalse',
              name  : 'interrupt',
              inputValue: "no",
              xtype : 'radiofield',
            }
          ]
        }*/
      ]
    }
  ]
},
{
  xtype : 'container',
  layout  : 'column',
  items : [
    {
      boxLabel  : _("Short␣Period␣OFF"),
      name    : 'type',
      inputValue  : 'off',
      id      : 'off',
      xtype   : 'radiofield',
      width   : 130
    },
    {
      xtype:'container',
      id:'offContainer',
      items: [
        {
          xtype : 'fieldcontainer',
          fieldLabel  : _("Slot␣length"),
          labelWidth: 120,
                items: [
                  {
                    id: 'offSlotLength',
                    name: 'slotLength',
                    xtype: 'dssTimeSelection',
                    type: 'length'
                  }
                ]
              },
              {
                xtype : 'fieldcontainer',
                fieldLabel  : _("OFF␣time"),
                labelWidth: 120,
                items: [
                  {
                    id: 'offOffTime',
                    name: 'offTime',
                    xtype: 'dssTimeSelection',
                    type: 'length'
                  }
                ]
              }
            ]
          }
        ]
      }
    ],
    buttons: [
      {
        text:_("Cancel"),
        id: 'btn-cancel'
      },
      {
        text: _("Save"),
        id: 'btn-save'
      }
    ]
  }
],


initComponent: function(){
  var me = this;
  me.callParent(arguments);
  me.initPage();
},

enableOff: function(enable){
  Ext.getCmp('offOffTime').enable(enable);
  Ext.getCmp('offSlotLength').enable(enable);
},

enableOn: function(enable){
  Ext.getCmp('onSlotLength').enable(enable);
  Ext.getCmp('onOnTime').enable(enable);
  if(enable){
    Ext.getCmp('onSingleSlot').enable();
    Ext.getCmp('onDetection').enable();
    Ext.getCmp('onDetection').setValue(false);
    Ext.getCmp('onDetection').setValue(true);
  }
  else{
    Ext.getCmp('onSingleSlot').disable();
    Ext.getCmp('onSingleSlotDate').disable();
    Ext.getCmp('onDetectionDevice').disable();
    Ext.getCmp('onDetection').disable();
    Ext.getCmp('onSingleSlotTime').enable(false);
  }
},

initPage: function(){
  var me = this;
  Ext.getCmp('btn-cancel').handler = function() {
    me.hide();
  };
  Ext.getCmp('btn-save').handler = function() {
    var form = Ext.getCmp('form').getForm();
    if(form.isValid()){
      var type = '';
      if(Ext.getCmp('on').getValue()){
        var length = Ext.getCmp('onSlotLength').getValue();
        var onTime = Ext.getCmp('onOnTime').getValue();
        type = 'on';
        if(onTime > length){
```

```
        Ext.Msg.alert(_("Error"), _("ON␣time␣has␣to␣be␣smaller␣than␣the␣slot␣length"));
        return;
      }
    }
    else if(Ext.getCmp('off').getValue()){
      var offTime = Ext.getCmp('offOffTime').getValue();
      var length = Ext.getCmp('offSlotLength').getValue();
      type = 'off';
      if(offTime > length){
        Ext.Msg.alert(_("Error"), _("OFF␣time␣has␣to␣be␣smaller␣than␣the␣slot␣length"));
        return;
      }
    }

    var data = form.getFieldValues();
    data = Ext.JSON.encode(data);
    me.setLoading(true);
    var event = Ext.create('DSS.json.Event', {name: 'smart-grid'});
    event.raise(
      {
        action_type: 'configDevice',
        deviceId: me.device.get('id'),
        config: data
      },
      {
        success: function(){
          var model = me.store.getById(me.device.get('id'));
          model.set('smartGridType', type);
          model.commit();
          me.setLoading(false);
          me.hide();
        },
        failure: function() {
          Ext.Msg.alert(_("Error"), _("Couldn\'t␣send␣event␣to␣DSS"));
          me.setLoading(false);
          me.hide();
        }
      }
    );
  }
};
Ext.getCmp('exclude').handler = function(){
  if(Ext.getCmp('exclude').getValue()){
    me.enableOff(false);
    me.enableOn(false);
  }
};
Ext.getCmp('on').handler = function(){
  if(Ext.getCmp('on').getValue()){
    me.enableOff(false);
    me.enableOn(true);
  }
};
Ext.getCmp('onSingleSlot').handler = function(){
  if(Ext.getCmp('onSingleSlot').getValue()){
    Ext.getCmp('onSingleSlotDate').enable();
    Ext.getCmp('onSingleSlotTime').enable(true);
  }
  else{
    Ext.getCmp('onSingleSlotDate').disable();
    Ext.getCmp('onSingleSlotTime').enable(false);
  }
};
Ext.getCmp('onDetection').handler = function(){
  if(Ext.getCmp('onDetection').getValue()){
    Ext.getCmp('onDetectionDevice').enable();
  }
  else{
    Ext.getCmp('onDetectionDevice').disable();
  }
}
Ext.getCmp('off').handler = function(){
  if(Ext.getCmp('off').getValue()){
    me.enableOff(true);
    me.enableOn(false);
  }
};

var iconTpl = Ext.create('Ext.Template', [
    // The pics are 16x16, +5 padding = 21
    '<div␣style="',
      '<tpl␣if="icon">',
            'background:left␣center␣no-repeat␣url(\'images/dss/{icon}\');',
          '</tpl>',
          'min-height:16;',
          'padding-left:21px;',
          '<tpl␣if="isPresent===false">color:gray;</tpl>',
        '">{text}</div>'
      ]
    );
    var combo = Ext.create('Ext.form.field.ComboBox', {
      fieldLabel: '',
      name: 'onDetectionDevice',
      id: 'onDetectionDevice',
      editable: false,
      store: me.store,
      queryMode: 'local',
      displayField: 'name',
      valueField: 'id',
      listConfig: { itemTpl: iconTpl },
      forceSelection: true
    });
    combo.on('render', function(thisBox) {
      // also create and render the picker on box rendering
      // otherwise render-time selection is not available
      var picker = thisBox.getPicker();
      picker.doAutoRender();
    });
    combo.on('select', function(field, value, options) {
      //display the icon
      var bg = 'background:none;';
      var icon = (value.length > 0 ? value[0].get('icon') : null);
      if (icon) {
        var url = 'images/dss/' + icon;
        bg = "background:left␣center␣no-repeat␣url('"+ url +"');";
      }
      field.setFieldStyle(bg + 'padding-left:21px;');
    });


    Ext.getCmp('startDetectionContainer').add(combo);
  },

  ajax: function(path, success){
    var me = this;
    Ext.Ajax.request({
      disableCaching: true,
      method: 'GET',
      timeout: 20000,
      url: '/json/' + path,
      success: success,
      failure: function(){
        Ext.Msg.alert(_("A␣terrible␣error␣happend.␣Aborting"));
        me.setLoading(false);
        me.hide();
      }
    });
  },

  openDevice: function(device){
    var me = this;
    me.device = device;
    me.setTitle(device.get('name'));
    me.show();
    me.setLoading(true);
    me.ajax("property/getString?path=/scripts/smart-grid/devices/" + device.get('id') + "/config/all", function(response){
      var data = Ext.JSON.decode(response.responseText);
      Ext.getCmp('onDetection').setValue(false);
      Ext.getCmp('onDetection').setValue(true);
      //Ext.getCmp('interruptTrue').setValue(true);
      Ext.getCmp('exclude').setValue(false);
      Ext.getCmp('exclude').setValue(true);
      if(data.ok){
        data = Ext.JSON.decode(data.result.value);
        Ext.getCmp('form').getForm().setValues(data);
        if(data.date !== undefined){
          var dt = new Date(data.date);
          Ext.getCmp('onSingleSlotDate').setValue(dt);
        }
        if(data.onDetectionDevice !== undefined){
          var box = Ext.getCmp('onDetectionDevice');
          box.fireEvent('select', box, [me.store.findRecord('id', data.onDetectionDevice)], null);
        }
      }
      me.setLoading(false);
    });
  }
});
```

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

B.2 Demonstration Energy Provider

```
package ch.ethz.baumachr.energyProvider;

import java.io.StringReader;
import java.nio.channels.SocketChannel;
import java.text.ParseException;
import java.util.Date;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.TreeSet;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;

public class Client {
  private final StringBuilder stringBuilder;
  private final DocumentBuilder domBuilder;
  private final HashMap<String, Device> devices;
  private final SocketChannel channel;
  private final DSSServer server;

  private final Logger LOG;

  public Client(DSSServer server, SocketChannel channel) throws
      ParserConfigurationException{
    this.channel = channel;
    this.server = server;

    stringBuilder = new StringBuilder();
    devices = new HashMap<String, Device>();

    DocumentBuilderFactory f = DocumentBuilderFactory.newInstance();
    domBuilder = f.newDocumentBuilder();

    LOG = SingletonUtil.instance().LOG;
  }

  public void newData(byte[] data){
    stringBuilder.append(new String(data));
    removeCommands();
  }

  private void newConfig(Document dom){
    boolean somethingChanged = false;
    LOG.log("", 5);
    LOG.log("New config:", 5);
    NodeList devicesNodeList = dom.getElementsByTagName("item");
    HashSet<String> newIds = new HashSet<String>();
    for(int i=0; i<devicesNodeList.getLength(); i++){
      HashMap<String, String> params = new HashMap<String, String>();
      NodeList paramNodes = devicesNodeList.item(i).getChildNodes();
      for(int j=0; j<paramNodes.getLength(); j++){
        Node param = paramNodes.item(j);
        params.put(param.getNodeName(), param.getTextContent());
      }
      if(params.containsKey("id")){
        String id = params.get("id");
        newIds.add(id);
        if(devices.containsKey(id)){
          Device old = devices.get(id);
          if(old.equals(params)){
            continue;
          }
        }
        somethingChanged = true;
        devices.put(id, Device.createDevice(params, this));
      }
      else{
        LOG.log("Received␣strange␣device␣spec␣without␣id", 0);
      }
    }

    //remove devices that are gone
    Iterator<String> it = devices.keySet().iterator();
    while(it.hasNext()){
      String id = it.next();
      if(!newIds.contains(id)){
        it.remove();
      }
    }


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

  private void newConsumptions(Document dom) {
    NodeList devicesNodeList = dom.getElementsByTagName("item");
    HashMap<Date, Double> newValues = new HashMap<Date, Double>();
    LOG.log("values␣" + devicesNodeList.getLength(), 20);
    for(int i=0; i<devicesNodeList.getLength(); i++){
      NodeList children = devicesNodeList.item(i).getChildNodes();
      Node timestamp = null;
      Node value = null;
      for(int j = 0; j<children.getLength(); j++){
        if(children.item(j).getNodeName() == "value"){
          value = children.item(j);
        }
        else if(children.item(j).getNodeName() == "timestamp"){
          timestamp = children.item(j);
        }
      }
      if(timestamp == null || value == null){
        continue;
      }
      String time = timestamp.getTextContent();
      try {
        Date date = SingletonUtil.instance().gmtDateFormat.parse(time);
        double v = Double.parseDouble(value.getTextContent());
        newValues.put(date, v);
        LOG.log(SingletonUtil.instance().formatTime(date) + "␣" + v, 20);
      } catch (ParseException e) {
        continue;
      }
    }
    this.server.newConsumptions(newValues, this);
  }

  public void send(String data){
    server.send(channel, data.getBytes());
  }

  private void removeCommands(){
    int first = stringBuilder.indexOf("<");
    for(int i = 0; i < first; i++){
      if( Character.isWhitespace(stringBuilder.charAt(0)) ){
        stringBuilder.deleteCharAt(0);
      }
      else{
        System.err.println("Received␣data␣not␣expected");
        server.closeConnection(channel);
        return;
      }
    }


    int last = stringBuilder.indexOf(">");
    String xml = null;
    if(last != -1){
      String root = stringBuilder.substring(1, last).trim();
      if(! root.endsWith("/")){
        last = stringBuilder.indexOf(root, last);
        if(last != -1){
          last = stringBuilder.indexOf(">", last);
        }
      }
```

```java
      if(last != -1){
        xml = stringBuilder.substring(0, last+1);
        stringBuilder.delete(0, last+1);
      }
    }
    if(xml != null){
      StringReader reader = new StringReader( xml );
      InputSource inputSource = new InputSource( reader );
      Document dom = null;
      try {
        dom = domBuilder.parse( inputSource );
      } catch (Exception e) {
        e.printStackTrace();
        server.closeConnection(channel);
        return;
      }

      String root = dom.getDocumentElement().getNodeName();
      if(root.equals("config")){
        newConfig(dom);
      }
      else if(root.equals("consumptions")){
        newConsumptions(dom);
      }
      else if(root.equals("time")){
        send("<time>" + SingletonUtil.instance().formatTime(new Date()) + "</time>\n");
      }
      else{
        LOG.log("Received␣unknown␣xml␣data", 0);
        LOG.log(xml, 1);
      }
      reader.close();
    }
  }

  public HashSet<ShortOffDevice> getShortOffDevices(){
    HashSet<ShortOffDevice> result = new HashSet<ShortOffDevice>();
    synchronized(devices){
      Iterator<Device> it = devices.values().iterator();
      while(it.hasNext()){
        Device device = it.next();
        if(device instanceof ShortOffDevice){
          ShortOffDevice d = (ShortOffDevice)device;
          result.add(d);
        }
      }
    }
    return result;
  }

  public TreeSet<DelayOnDevice> getDelayOnDevices(){
    TreeSet<DelayOnDevice> result = new TreeSet<DelayOnDevice>(SingletonUtil.instance().delayOnComparator);
    synchronized(devices){
      Iterator<Device> it = devices.values().iterator();
      while(it.hasNext()){
        Device device = it.next();
        if(device instanceof DelayOnDevice){
          DelayOnDevice d = (DelayOnDevice)device;
          result.add(d);
        }
      }
    }
    return result;
  }

  public String getId() {
    return this.channel.socket().getRemoteSocketAddress().toString().substring(1);
  }
}
```

Listing B.5: Client.java

```java
package ch.ethz.baumachr.energyProvider;

import java.io.IOException;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.HashMap;
import java.util.Observable;
import java.util.Observer;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class Controller implements Runnable, Observer{
  public static final int LOG_LEVEL = 10;

  private final DSSServer server;
  private final IntradayMarketDataPoint[] data;
  private final HashMap<Date, Double> dataMapping;

  private DocumentBuilder domBuilder;

  private final Logger LOG;

  private boolean doScheduling = false;

  public Controller(DSSServer s) {
    LOG = SingletonUtil.instance().LOG;
    this.server = s;
    data = new IntradayMarketDataPoint[48];
    dataMapping = new HashMap<Date, Double>();
    try {
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      dbf.setNamespaceAware(false);
      dbf.setValidating(false);
      dbf.setFeature("http://xml.org/sax/features/namespaces", false);
      dbf.setFeature("http://xml.org/sax/features/validation", false);
      dbf.setFeature("http://apache.org/xml/features/nonvalidating/load-dtd-grammar",
          false);
      dbf.setFeature("http://apache.org/xml/features/nonvalidating/load-external-dtd",
          false);
      domBuilder = dbf.newDocumentBuilder();
    } catch (ParserConfigurationException e) {
      e.printStackTrace();
    }

    //add myself to receive update calls from the server on config change
    server.addObserver(this);
  }

  private boolean getMarketData(){
    boolean nothingChanged = true;
    try {
      //TODO change in data layout
      if(false){
      Document dom = domBuilder.parse("http://www.epexspot.com/en/market-data/intraday
          /intraday-table/-/DE");
      NodeList rows = dom.getElementsByTagName("tr");
      System.out.println(rows.getLength());
      for(int i = 3; i < rows.getLength() - 2; i++){
        NodeList columns = rows.item(i).getChildNodes();
        int index = 0;
        int columnIndex = i - 3;
        if(data[columnIndex] == null){
          Calendar cal = new GregorianCalendar();
          cal.setTime(new Date());
          cal.set(Calendar.HOUR_OF_DAY, columnIndex);
          cal.set(Calendar.MINUTE, 30);
          cal.set(Calendar.SECOND, 0);
          cal.set(Calendar.MILLISECOND, 0);

          data[columnIndex] = new IntradayMarketDataPoint();
          data[columnIndex].setHour(cal.getTime());

          cal.add(Calendar.DAY_OF_MONTH, 1);
          data[columnIndex + 24] = new IntradayMarketDataPoint();
          data[columnIndex + 24].setHour(cal.getTime());
        }
        IntradayMarketDataPoint today = data[columnIndex];
        IntradayMarketDataPoint tomorow = data[columnIndex + 24];
        for(int j = 0; j < columns.getLength(); j++){
          Node cellNode = columns.item(j);
          if(cellNode.getNodeType() == Node.ELEMENT_NODE){
            String cell = cellNode.getTextContent().trim();

            IntradayMarketDataPoint column = tomorow;
            switch(index){
              case 0: //hour (01-02)
                break;
              case 1: //low today
              case 8: //low tomorrow
                break;
              case 2: //high today
              case 9: //high tomorrow
                break;
              case 3:  //last today
                column = today;
                //break trough intended
              case 10: //last tomorrow
                try{
                  double last = Double.parseDouble(cell);
                  if(!column.isSameLast(last)){
                    LOG.log("Last␣for␣" +
                        SingletonUtil.instance().formatTime(column.getDate()) +
                        "␣changed␣from␣" + column.getLast() + "␣to␣" + cell, 15);
                    column.setLast(Double.parseDouble(cell));
                    nothingChanged = false;
                  }
                }
                catch(NumberFormatException e){}
                break;
              case 4: //avg
                break;
            }
            index++;
          }
        }
      }
      if(!nothingChanged){
        LOG.log("Market␣data␣changed", 8);
        for(int i = 0; i < data.length; i++){
          if(data[i].available()){
            dataMapping.put(data[i].getDate(), data[i].getLast());
          }
        }
        server.newMarketData(dataMapping);
      }
      }
    } catch (SAXException e2) {
      e2.printStackTrace();
    } catch (IOException e2) {
      e2.printStackTrace();
    }
    return !nothingChanged;
  }

  private void scheduleDevices(){
    /*LOG.log("Doing scheduling", 6);
    TreeSet<DelayOnDevice> devices = server.getDelayOnDevices();
    Iterator<DelayOnDevice> it = devices.iterator();

    Calendar startTime = Calendar.getInstance();
    //settle down first
    startTime.add(Calendar.SECOND, 600);
    while(it.hasNext()){
      DelayOnDevice dev = it.next();
      Calendar deadline = dev.getDeadline();
      if(startTime.after(deadline)){
        dev.turnOn(deadline.getTime());
      }
      else{
        dev.turnOn(startTime.getTime());
        startTime.add(Calendar.SECOND, dev.getOnTime());
      }
    }*/
  }

  @Override
  public void run() {
    Calendar nextMarketUpdate = Calendar.getInstance();
    while(true){
      try {
        boolean schedule = false;
        synchronized(this){
          if(doScheduling) {
            doScheduling = false;
            schedule = true;
          }
        }
        if(schedule) scheduleDevices();
```

```java

        Calendar now = Calendar.getInstance();
        long dt = nextMarketUpdate.getTimeInMillis() - now.getTimeInMillis();
        boolean marketChanged = false;
        if(dt <= 0){
          LOG.log("Updateing␣market␣data", 18);
          int minutes = 5;
          marketChanged = getMarketData();
          nextMarketUpdate = now;
          nextMarketUpdate.add(Calendar.MINUTE, minutes);
          dt = minutes * 60 * 1000;
        }

        LOG.log("Waiting␣" + dt/1000 + "s", 25);
        synchronized(this){
          if(marketChanged){
            doScheduling = true;
          }
          if(!doScheduling){
            this.wait(dt);
          }
        }
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }

  @Override
  public void update(Observable arg0, Object arg1) {
    synchronized(this){
      this.doScheduling = true;
      this.notify();
    }
  }
}
```

Listing B.6: Controller.java

```java
package ch.ethz.baumachr.energyProvider;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public abstract class Device {
  public static final int LOG_LEVEL = 1;
  protected final String id;
  protected final HashMap<String, String> params;
  protected final Client client;

  protected final Logger LOG;


  public Device(final HashMap<String, String> params, Client c){
    this.id = params.get("id");
    this.params = params;
    this.client = c;
    this.LOG = SingletonUtil.instance().LOG;
    print();
  }

  public static Device createDevice(HashMap<String, String> params, Client c){
    String type = params.get("type");
    if(type == null){
      return null;
    }
    try {
      if(type.equals("on")){
        return new DelayOnDevice(params, c);
      }
      else if(type.equals("off")){
        return new ShortOffDevice(params, c);
      }
    }
    catch (WrongParameterException e) {
      e.printStackTrace();
      return null;
    }
    return null;
  }

  protected void checkParams(String type, String[] keys) throws
      WrongParameterException{
    if(!params.containsKey("type")){
      throw new WrongParameterException();
    }
    if(!params.get("type").equals(type)){
      throw new WrongParameterException();
    }
    for(int i=0; i<keys.length; i++){
      if(!params.containsKey(keys[i])){
        throw new WrongParameterException();
      }
    }
  }

  public void print(){
    LOG.log("Device␣" + id, 5);
    Iterator<Map.Entry<String,String>> it = params.entrySet().iterator();
    while (it.hasNext()) {
      Map.Entry<String,String> pairs = it.next();
      LOG.log("\t" + pairs.getKey() + "␣=␣" + pairs.getValue(), 5);
    }
  }

  public class WrongParameterException extends Exception {
    private static final long serialVersionUID = 1L;
  }

  public String getType(){
    return params.get("type");
  }

  public boolean equals(HashMap<String, String> otherParams){
    if(params.size() != otherParams.size()){
      return false;
    }
    Iterator<String> it = params.keySet().iterator();
    while(it.hasNext()){
      String p = it.next();
      if(!otherParams.containsKey(p)){
        return false;
      }
      else if(!otherParams.get(p).equals(params.get(p))){
        return false;
      }
    }
    return true;
  }
}
```

Listing B.7: Device.java

```java
package ch.ethz.baumachr.energyProvider;

import java.text.ParseException;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.HashMap;

public class DelayOnDevice extends Device {

  public static final int LOG_LEVEL = 10;

  private final Logger LOG = SingletonUtil.instance().LOG;

  private Date scheduledAt = null;

  private final String[] keys = {
    //"startDetection",
    //"interrupt",
    "onTime",
    "slotLength",
    "startTime"
  };

  public DelayOnDevice(HashMap<String, String> params, Client c) throws
      WrongParameterException {
    super(params, c);
    checkParams("on", keys);
  }

  public Date getStartTime(){
    try {
      return SingletonUtil.instance().gmtDateFormat.parse(params.get("startTime"));
    } catch (ParseException e) {
      e.printStackTrace();
    }
    return null;
  }

  public Calendar getDeadline() {
    Calendar start = new GregorianCalendar();
    start.setTime(getStartTime());
    start.add(Calendar.SECOND, Integer.parseInt(params.get("slotLength")) - Integer.parseInt(params.get("onTime")));
    return start;
  }

  public int getOnTime(){
    return Integer.parseInt(params.get("onTime"));
  }

  public Date getScheduledTime(){
    return scheduledAt;
  }

  public void turnOn(Date time){
    if(scheduledAt == null){
      reschedule(time);
    }
  }

  public void reschedule(Date time){
    String at = (time != null)?"␣at='" + SingletonUtil.instance().formatTime(time) + "'":"";
    client.send("<delayon" + at + ">" + id + "</delayon>\n");
    LOG.log("Turning␣on␣device␣" + id, 4);
    LOG.log("\t@" + ((time==null)?"now":SingletonUtil.instance().formatTime(time)), 4);
    scheduledAt = time;
  }
}
```

Listing B.8: DelayOnDevice.java

```
package ch.ethz.baumachr.energyProvider;

import java.util.HashMap;

public class ShortOffDevice extends Device {

  private final String[] keys = {
    "offTime",
    "slotLength"
  };

  public ShortOffDevice(HashMap<String, String> params, Client c) throws WrongParameterException {
    super(params, c);
    checkParams("off", keys);
  }

  public void turnOff(int seconds){
    String s = (seconds > 0)?"␣seconds='" + seconds + "'":"";
    client.send("<shortoff" + s + ">" + id + "</shortoff>\n");
  }

  public void turnOn(){
    client.send("<shortoff␣seconds='0'>" + id + "</shortoff>\n");
  }
}
```

Listing B.9: ShortOffDevice.java

```
package ch.ethz.baumachr.energyProvider;


import java.io.IOException;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.nio.ByteBuffer;
```

```java
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.ServerSocketChannel;
import java.nio.channels.SocketChannel;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Observable;
import java.util.Set;
import java.util.TreeSet;

import javax.xml.parsers.ParserConfigurationException;

public class DSSServer extends Observable implements Runnable {
  public static final int LOG_LEVEL = 10;

  private final ServerSocketChannel serverSocket;
  private final Selector selector;
  private ByteBuffer reusableBuffer;

  private EnergyProvider gui;

  private List<SocketChannel> clients;

  private HashMap<SocketChannel, List<ByteBuffer>> writeRequests;

  private final Logger LOG;

  public DSSServer(int port, EnergyProvider gui){
    this.gui = gui;
    LOG = SingletonUtil.instance().LOG;
    reusableBuffer = ByteBuffer.allocate(256);
    ServerSocketChannel tempServerSocket = null;
    Selector tempSelector = null;
    try {
      tempServerSocket = ServerSocketChannel.open();
      tempSelector = Selector.open();
      tempServerSocket.configureBlocking(false);
      tempServerSocket.socket().bind(new InetSocketAddress(port));
      tempServerSocket.register(tempSelector, SelectionKey.OP_ACCEPT);
    } catch (IOException e) {
      e.printStackTrace();
    }
    this.serverSocket = tempServerSocket;
    selector = tempSelector;

    clients = new ArrayList<SocketChannel>();

    writeRequests = new HashMap<SocketChannel, List<ByteBuffer>>();
  }

  public void send(SocketChannel socket, byte[] data) {
    sendIt(socket, data);

    // Finally, wake up our selecting thread so it can make the required changes
    this.selector.wakeup();
  }

  private void sendIt(SocketChannel socket, byte[] data) {
    synchronized (this.writeRequests) {
      List<ByteBuffer> queue = this.writeRequests.get(socket);
      if(queue == null){
        queue = new ArrayList<ByteBuffer>();
        this.writeRequests.put(socket, queue);
      }
      queue.add(ByteBuffer.wrap(data));
    }
  }

  public HashSet<ShortOffDevice> getShortOffDevices(){
    HashSet<ShortOffDevice> result = new HashSet<ShortOffDevice>();
    synchronized(clients){
      Iterator<SocketChannel> it = clients.iterator();
      while(it.hasNext()){
        SocketChannel channel = it.next();
        Client c = (Client)channel.keyFor(selector).attachment();
        HashSet<ShortOffDevice> l = c.getShortOffDevices();
        result.addAll(l);
      }
    }
    return result;
  }

  public TreeSet<DelayOnDevice> getDelayOnDevices(){
    TreeSet<DelayOnDevice> result = new TreeSet<DelayOnDevice>(SingletonUtil.instance().delayOnComparator);
    synchronized(clients){
      Iterator<SocketChannel> it = clients.iterator();
      while(it.hasNext()){
        SocketChannel channel = it.next();
        Client c = (Client)channel.keyFor(selector).attachment();
        TreeSet<DelayOnDevice> l = c.getDelayOnDevices();
        result.addAll(l);
      }
    }
    return result;
  }

  @Override
  public void run() {
    try {
      while (!Thread.interrupted()) {

        // iterate through the writeRequests List and mark clients that want to write
        synchronized(this.writeRequests) {
          Iterator<SocketChannel> writeRequestKeys = this.writeRequests.keySet().iterator();
          while (writeRequestKeys.hasNext()) {
            SocketChannel c = writeRequestKeys.next();
            if(!writeRequests.get(c).isEmpty()){
              c.keyFor(selector).interestOps(SelectionKey.OP_WRITE);
            }
          }
        }

        //TODO solve this in a better way
        //this does not work if every 9 seconds another client connects
        // to the select call with a timeout
        selector.select(10000);

        Set<SelectionKey> selected = selector.selectedKeys();

        // check if the timeout of select expired or if there is a action going on
        if(selected.isEmpty()){
          // send a ping message to all clients
          synchronized(clients){
            Iterator<SocketChannel> it = clients.iterator();
            while(it.hasNext()){
              SocketChannel c = it.next();
              sendIt(c, "<ping/>\n".getBytes());
            }
          }
          continue;
        }

        // analyze all selected keys
        Iterator<SelectionKey> itr = selected.iterator();
        while (itr.hasNext()){
          SelectionKey key = itr.next();
          if(key.isAcceptable()){
            accept(key);
          }
          else if(key.isReadable()){
            read(key);
          }
          else if(key.isWritable()){
            write(key);
          }
          else{
            System.err.println("strange");
          }
        }
        // clear the keys from the set since they are already processed
        selected.clear();
      }
    }
    catch(Exception ex){
      ex.printStackTrace();
    }
  }

  private void accept(SelectionKey key) throws IOException, ParserConfigurationException{
    SocketChannel c = serverSocket.accept();
    Socket s = c.socket();
```

```java
    LOG.log(s.getInetAddress().getHostAddress() + ":" + s.getPort() + "␣connected", 3);
    c.configureBlocking(false);
    SelectionKey sk = c.register(selector, SelectionKey.OP_READ);
    Client client = new Client(this, c);

    // add the client to the local clients List
    synchronized(clients){
      clients.add(c);
    }
    // attach the client to the key
    sk.attach(client);

    gui.connected(client.getId());
  }

  private void read(SelectionKey key) throws ParserConfigurationException{
    SocketChannel c = (SocketChannel)key.channel();
    reusableBuffer.clear();
    int bytesRead = 0;
    try {
      bytesRead = c.read(reusableBuffer);
    } catch (IOException e) {
      closeConnection(c);
      return;
    }
    if(bytesRead < 0){
      closeConnection(c);
      return;
    }
    if(bytesRead > 0){
      Client client = (Client) key.attachment();
      byte[] substring = new byte[bytesRead];
      reusableBuffer.flip();
      reusableBuffer.get(substring);
      client.newData(substring);
    }
  }

  public void closeConnection(SocketChannel c){
    SelectionKey sk = c.keyFor(selector);
    Client clientObject = (Client)sk.attachment();
    gui.disconnected(clientObject.getId());
    Socket s = c.socket();
    LOG.log(s.getInetAddress().getHostAddress() + ":" + s.getPort() + "␣disconnected", 3);
    synchronized(writeRequests){
      writeRequests.remove(c);
    }
    clients.remove(c);
    try {
      c.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
    return;
  }

  private void write(SelectionKey key) throws IOException, ParserConfigurationException{
    SocketChannel c = (SocketChannel)key.channel();
    synchronized(writeRequests){
      List<ByteBuffer> list = writeRequests.get(c);
      while(!list.isEmpty()){
        ByteBuffer buf = list.get(0);
        c.write(buf);
        if (buf.remaining() > 0) {
          // socket's buffer fills up
          break;
        }
        list.remove(0);
      }
      if (list.isEmpty()) {
        // We wrote away all data, so we're no longer interested
        // in writing on this socket. Switch back to waiting for
        // data.
        key.interestOps(SelectionKey.OP_READ);
      }
    }
  }

  public void newConsumptions(HashMap<Date, Double> newValues, Client c) {
    gui.addConsumptionValues(newValues, c.getId());
  }

  public void newMarketData(HashMap<Date, Double> dataMapping){
    gui.newMarketData(dataMapping);
  }

  public void newConfig(){
    this.setChanged();
    this.notifyObservers();
    this.clearChanged();
  }
}
```

Listing B.10: DSSServer.java

```
package ch.ethz.baumachr.energyProvider;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.DisplayMode;
import java.awt.GraphicsDevice;
import java.awt.GraphicsEnvironment;
import java.awt.MouseInfo;
import java.awt.Point;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Calendar;
import java.util.Date;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.TreeSet;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.axis.ValueAxis;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.time.Millisecond;
import org.jfree.data.time.Second;
import org.jfree.data.time.TimeSeries;
import org.jfree.data.time.TimeSeriesCollection;
import org.jfree.ui.ApplicationFrame;

public class EnergyProvider extends ApplicationFrame {
  private static final long serialVersionUID = 1L;

  private final DSSServer dSSserver;
  private final TimeSeriesCollection consumptionDataset;
  private final TimeSeries consumptionSum;
  private final HashMap<String, TimeSeries> consumptionSeries;
  private final HashMap<String, Long> consumptionAlreadyAddedUntil;
  private TimeSeries marketSeries;
  private final Logger LOG;

  final private ChartPanel consumptionChartPanel;

  public EnergyProvider(String title) {
    super(title);

    LOG = SingletonUtil.instance().LOG;

    JButton underrun = new JButton("Underrun");
    underrun.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        TreeSet<DelayOnDevice> delayOns = dSSserver.getDelayOnDevices();
        Iterator<DelayOnDevice> onIt = delayOns.iterator();
        while(onIt.hasNext()){
          DelayOnDevice device = onIt.next();
          device.reschedule(null);
        }
        HashSet<ShortOffDevice> shortOffs = dSSserver.getShortOffDevices();
        Iterator<ShortOffDevice> offIt = shortOffs.iterator();
```

```java
        while(offIt.hasNext()){
          ShortOffDevice device = offIt.next();
          LOG.log("Turning off device " + device.id, 4);
          device.turnOn();
        }
      }
    });

    JButton overrun = new JButton("Overrun");
    overrun.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        HashSet<ShortOffDevice> devices = dSSserver.getShortOffDevices();
        Iterator<ShortOffDevice> it = devices.iterator();
        while(it.hasNext()){
          ShortOffDevice device = it.next();
          LOG.log("Turning off device " + device.id, 4);
          device.turnOff(0);
        }
      }
    });

    consumptionSeries = new HashMap<String, TimeSeries>();
    consumptionAlreadyAddedUntil = new HashMap<String, Long>();
    consumptionSum = new TimeSeries("Sum");
    consumptionSum.setMaximumItemCount(400);
    consumptionDataset = new TimeSeriesCollection(consumptionSum);
    JFreeChart chart = ChartFactory.createTimeSeriesChart("Energy Consumption", //title
      "Time", //label of x-axis
      "Energy consumption [W]", //label of y-axis
      consumptionDataset,
      true, //legend
      true, //tooltips
      false //urls
    );
    consumptionChartPanel = new ChartPanel( createTimeSeriesChart(chart, 5 * 60 * 1000, false) );
    consumptionChartPanel.setPreferredSize( new Dimension( 600, 250 ));

    /*this.marketSeries = new TimeSeries("Market");
    TimeSeriesCollection dataset = new TimeSeriesCollection(this.marketSeries);
    chart = ChartFactory.createTimeSeriesChart("Intraday Market", //title
      "Time", //label of x-axis
      "Cost [Euro/kWh]", //label of y-axis
      dataset,
      false, //legend
      true, //tooltips
      false //urls
    );
    ChartPanel marketChartPanel = new ChartPanel( createTimeSeriesChart(chart, 24 * 60 * 60 * 1000, null, true) );
    marketChartPanel.setPreferredSize( new Dimension( 600, 250 ) );*/


    JPanel pu = new JPanel();
    pu.add(underrun);

    JPanel po = new JPanel();
    po.add(overrun);


    JPanel mainPanel = new JPanel();
    mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));
    mainPanel.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
    //mainPanel.add(marketChartPanel);
    mainPanel.add(consumptionChartPanel);
    mainPanel.add(pu);
    mainPanel.add(po);


    this.setResizable(false);
    this.getContentPane().add(mainPanel, BorderLayout.CENTER);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //adjust window size to the content.
    this.pack();
    positionAtCenterOfActiveScreen();


    //start other threads
    dSSserver = new DSSServer(50007, this);
    Thread dSSthread = new Thread(dSSserver);
    dSSthread.start();
    Controller controller = new Controller(dSSserver);
    Thread controllerThread = new Thread(controller);
    controllerThread.start();
  }

  /**
  * Creates a sample chart.
  *
  * @param dataset  the dataset.
  *
  * @return A sample chart.
  */
  private JFreeChart createTimeSeriesChart(JFreeChart chart, double xRange, boolean shapes) {
    final XYPlot plot = chart.getXYPlot();
    //plot.setBackgroundPaint(Color.lightGray);
    plot.setDomainGridlinePaint(Color.white);
    plot.setRangeGridlinePaint(Color.white);

    final XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer();
    renderer.setBaseLinesVisible(true);
    //renderer.setSeriesLinesVisible(0, true);
    renderer.setBaseShapesVisible(shapes);
    //renderer.setSeriesShapesVisible(0, shapes);
    plot.setRenderer(renderer);

    // change the auto tick unit selection to integer units only...
    final ValueAxis rangeX = plot.getDomainAxis();
    //rangeX.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
    //rangeX.setRange(0, 99);
    rangeX.setAutoRange(true);
    rangeX.setFixedAutoRange(xRange);  // 5 minutes

    final NumberAxis rangeY = (NumberAxis) plot.getRangeAxis();
    rangeY.setStandardTickUnits(NumberAxis.createStandardTickUnits());

    //rangeY.setAutoRange(true);

    //chart.setBackgroundPaint(Color.white);
    return chart;
  }

  private void positionAtCenterOfActiveScreen(){
    //compute the center of the screen and move the window there
    Point mouse  = MouseInfo.getPointerInfo().getLocation();
    GraphicsDevice[] gds = GraphicsEnvironment.getLocalGraphicsEnvironment().getScreenDevices();

    //hack to always open the window on my second screen
    //mouse.y = 1200;
    //mouse.x = 10;

    int screen = 0;
    int h = 0;
    int w = 0;
    for(int i=0; i<gds.length; i++){
      DisplayMode m = gds[i].getDisplayMode();
      int tempW = m.getWidth() + w;
      int tempH = m.getHeight() + h;
      if(mouse.x < tempW && mouse.y < tempH){
        screen = i;
        break;
      }
      if(mouse.x >= tempW){
        w += m.getWidth();
      }
      if(mouse.y >= tempH){
        h += m.getHeight();
      }
    }
    LOG.log("Screen " + screen + " width " + w + " height " + h, 10);

    Point p = new Point();
    GraphicsDevice gd = gds[screen];
    int screenWidth = gd.getDisplayMode().getWidth();
    int screenHeight = gd.getDisplayMode().getHeight();
    int hight = this.getHeight();
    int width = this.getWidth();
    double centreX = screenWidth / 2 + w;
    double centreY = screenHeight / 2 + h;
    p.setLocation(centreX-width/2, centreY-hight/2);
    this.setLocation(p);
  }

  public void addConsumptionValues(HashMap<Date, Double> newValues, String id) {
    final TreeSet<Date> keys = new TreeSet<Date>(newValues.keySet());
    synchronized(consumptionSeries){
      TimeSeries clientsSeries= null;
      if(!consumptionSeries.containsKey(id)){
        clientsSeries = new TimeSeries(id);
        clientsSeries.setMaximumItemCount(400);
        consumptionSeries.put(id, clientsSeries);
        consumptionDataset.addSeries(clientsSeries);
      }
      else{
        clientsSeries = consumptionSeries.get(id);
      }

      Iterator<Date> it = keys.iterator();

      while(it.hasNext()){
        Date date = it.next();
        //hack to remove the newest element (because of dss bug #1299)
        if(!it.hasNext()){
          break;
        }
        Calendar cal = Calendar.getInstance();
        cal.setTime(date);
        long seconds = cal.getTimeInMillis() / 1000;
        double consumption = newValues.get(date);
        Second s = new Second(date);
        if(clientsSeries.getValue(s) == null){
          clientsSeries.addOrUpdate(new Second(date), consumption);
        }

        long startAt = 0;
        if(consumptionAlreadyAddedUntil.containsKey(id)){
          startAt = consumptionAlreadyAddedUntil.get(id);
        }
        if(startAt == 0 || startAt <= seconds){
          if(startAt == 0){
            startAt = seconds;
          }
          for(long i = startAt; i <= seconds; i++){
            cal.setTimeInMillis(i * 1000);
            s = new Second(cal.getTime());
            Number v = consumptionSum.getValue(s);
            double value = (v == null)?0:v.doubleValue();
            consumptionSum.addOrUpdate(s, value + consumption);
          }
          consumptionAlreadyAddedUntil.put(id, seconds + 1);
        }
      }
    }
    consumptionChartPanel.updateUI();
  }

  public void newMarketData(HashMap<Date, Double> newValues) {
    synchronized(marketSeries){
      Iterator<Date> it = newValues.keySet().iterator();
      while(it.hasNext()){
        Date date = it.next();
        double cost = newValues.get(date);
        marketSeries.addOrUpdate(new Millisecond(date), cost);
      }
    }
  }

  public static void main(String[] args) {
    final EnergyProvider energyProvider = new EnergyProvider("Smart Grid Control");
    //display the window
    energyProvider.setVisible(true);
  }

  public void connected(String id){
  }

  public void disconnected(String id) {
    if(consumptionSeries.containsKey(id)){
      consumptionDataset.removeSeries(consumptionSeries.get(id));
      consumptionSeries.remove(id);
      consumptionAlreadyAddedUntil.remove(id);
    }
  }
}
```

Listing B.11: EnergyProvider.java

```java
package ch.ethz.baumachr.energyProvider;

import java.lang.reflect.Field;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Logger {

  public final int LEVEL = 1;

  private final String className;

  public Logger(){
    className = this.getClass().getCanonicalName();
  }

  public synchronized void log(final String s, int level){
    int LOG_LEVEL = LEVEL;
    String callerName = "";
    String methodName = "";
    int lineNumber = 0;
    try {
      throw new Exception();
    }
    catch(Exception e){
      StackTraceElement[] trace = e.getStackTrace();
      for(int i=0; i<trace.length; i++){
        if(!className.equals(trace[i].getClassName())){
          String fullCallerName = trace[i].getClassName();
          String[] callerClassNameArray = fullCallerName.split("\\.");
          callerName = callerClassNameArray[callerClassNameArray.length - 1];
          methodName = trace[i].getMethodName();
          lineNumber = trace[i].getLineNumber();

          try {
            Field f = Class.forName(fullCallerName).getField("LOG_LEVEL");
            LOG_LEVEL = f.getInt(null);
          } catch (Exception e1) {}

          break;
        }
      }
    }
    if(level < LOG_LEVEL){
      //[2011-10-27 09:20:36] [poll:441]
      final SimpleDateFormat sdf = new SimpleDateFormat("[yyyy-MM-dd HH:mm:ss] ");
      final Date d = new Date();
      System.out.print(sdf.format(d));
      System.out.print("[" + callerName + "." + methodName + ":" + lineNumber + "] ");
      System.out.println(s);
    }
  }
}
```

Listing B.12: Logger.java

```java
package ch.ethz.baumachr.energyProvider;

import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Comparator;
import java.util.Date;
import java.util.Locale;
import java.util.TimeZone;

public class SingletonUtil {

  private static SingletonUtil instance = null;

  public static SingletonUtil instance(){
    if(instance == null){
      instance = new SingletonUtil();
    }
    return instance;
  }

  public final SimpleDateFormat gmtDateFormat;
  public Logger LOG;
  public final Comparator<DelayOnDevice> delayOnComparator;

  private SingletonUtil(){
    gmtDateFormat = new SimpleDateFormat("EEE, d MMM yyyy HH:mm:ss z", Locale.ENGLISH); //Wed, 12 Oct 2011 07:12:00 GMT
    gmtDateFormat.setTimeZone(TimeZone.getTimeZone("GMT"));
    LOG = new Logger();

    delayOnComparator = new Comparator<DelayOnDevice>() {
      @Override
      public int compare(DelayOnDevice d1, DelayOnDevice d2) {
        Calendar c1 = d1.getDeadline();
        Calendar c2 = d2.getDeadline();
        if(c1.before(c2)) return -1;
        if(c1.after(c2)) return 1;
        return 0;
      }
    };
  }

  public String formatTime(Date time){
    return gmtDateFormat.format(time);
  }
}
```

Listing B.13: SingletonUtil.java

```
package ch.ethz.baumachr.energyProvider;

import java.util.Date;

public class IntradayMarketDataPoint {
  private Date date;
  private double last;
  private boolean lastSet;

  public IntradayMarketDataPoint(){
    lastSet = false;
  }

  public Date getDate(){
    return date;
  }
  public void setHour(Date d) {
    date = d;
  }

  public boolean available(){
    return lastSet;
  }

  public void setLast(double l){
    this.last = l;
    this.lastSet = true;
  }
  public double getLast(){
    return last;
  }
  public boolean isSameLast(double l){
    return l==last;
  }
}
```

Listing B.14: IntradayMarketDataPoint.java

B.3 Simulation

```
import random

class Device(object):
  def __init__(self):
    self.on = random.choice([True, False])
    self.consumption = random.randint(0, 2000)

  def isOn(self, time):
    r = random.random()
    if r > 0.9:
      self.on = not self.on
    return self.on

  def shutOff(self, time):
    pass

  def powerOn(self, time):
    pass

  def getConsumption(self, time):
    return self.consumption if self.isOn(time) else 0


class ShortOff(Device):
  def __init__(self):
    super(ShortOff, self).__init__()
    self.on = True
    self.offTime = random.randint(60, 60 * 60 * 4)
    self.slotLength = random.randint(self.offTime, 60 * 60 * 24)
    self.alreadyOffInSlot = 0
    self.slotStart = 0
    self.lastOffInSlot = 0

  def isOn(self, time):
    if not self.on:
      offSinceLastOff = (time - self.lastOffInSlot)
      if self.offTime - (self.alreadyOffInSlot + offSinceLastOff) < 0:
        self.alreadyOffInSlot += offSinceLastOff
        self.on = True
    return self.on

  def shutOff(self, time):
    if self.on:
      slotEnd = self.slotStart + self.slotLength
      if slotEnd < time:
        self.alreadyOffInSlot = 0
        self.slotStart = time
        self.lastOffInSlot = time
        self.on = False
      elif self.alreadyOffInSlot < self.offTime:
        self.lastOffInSlot = time
        self.on = False

  def shutOffPossible(self, time):
    slotEnd = self.slotStart + self.slotLength
    return self.on and (slotEnd < time or self.alreadyOffInSlot < self.offTime)

  def powerOn(self, time):
    if not self.on:
      offSinceLastOff = (time - self.lastOffInSlot)
      self.on = True
      self.alreadyOffInSlot += offSinceLastOff

class DelayOn(Device):
  def __init__(self):
    super(DelayOn, self).__init__()
    self.slotStartInterval = 60 * 60 * 24
    self.slotStart = random.randint(0, self.slotStartInterval)
    self.onTime = random.randint(60, 60 * 60 * 4)
    self.slotLength = random.randint(self.onTime, 60 * 60 * 8)

    self.actualStart = self.slotStart + self.slotLength - self.onTime

  def isOn(self, time):
    if time < self.slotStart + self.slotLength - self.onTime:
      return (self.actualStart <= time) and (time < self.actualStart + self.onTime)
    else:
      self.actualStart = time
      self.slotStart += self.slotLength + random.randint(0, self.slotStartInterval)
      return True

  def powerOn(self, time):
    if self.powerOnPossible(time):
      self.actualStart = time
      self.slotStart += self.slotLength + random.randint(0, self.slotStartInterval)

  def powerOnPossible(self, time):
    return (self.slotStart <= time) and (time <= self.slotStart + self.slotLength -
        self.onTime)

  def schedulable(self, time):
    return time > self.slotStart

"""d = DelayOn()
print d.slotStart, d.slotLength, d.onTime
```

```


previous = -1
for t in xrange(0, 60 * 60 * 24 * 7, 10):
  temp = d.getConsumption(t)
  d.powerOn(t)
  if previous != temp:
    print t, d.getConsumption(t)
    previous = temp"""
```

Listing B.15: device.py

```
#!/usr/bin/python

import random
from multiprocessing import Process, Queue
import sys
from device import Device, ShortOff, DelayOn


numberOfDevices = 200000

factor = 1730
timestep = 60
points = [
  ( 0 * 60 * 60, 49669.6409090909 * factor),
  ( 1 * 60 * 60, 48519.3590909091 * factor),
  ( 2 * 60 * 60, 47920.1045454546 * factor),
  ( 3 * 60 * 60, 47784.0590909091 * factor),
  ( 4 * 60 * 60, 48360.1181818182 * factor),
  ( 5 * 60 * 60, 50001.2045454546 * factor),
  ( 6 * 60 * 60, 54132.3454545455 * factor),
  ( 7 * 60 * 60, 59275.2136363636 * factor),
  ( 8 * 60 * 60, 61325.8181818182 * factor),
  ( 9 * 60 * 60, 62002.9818181818 * factor),
  (10 * 60 * 60, 63282.1454545454 * factor),
  (11 * 60 * 60, 64865.1090909091 * factor),
  (12 * 60 * 60, 64989.1181818182 * factor),
  (13 * 60 * 60, 64551.2727272727 * factor),
  (14 * 60 * 60, 63373.5590909091 * factor),
  (15 * 60 * 60, 62300.2000000000 * factor),
  (16 * 60 * 60, 62997.7681818182 * factor),
  (17 * 60 * 60, 65050.3954545455 * factor),
  (18 * 60 * 60, 65487.3136363636 * factor),
  (19 * 60 * 60, 64111.5318181818 * factor),
  (20 * 60 * 60, 61880.1090909091 * factor),
  (21 * 60 * 60, 58534.3909090909 * factor),
  (22 * 60 * 60, 56449.8636363636 * factor),
  (23 * 60 * 60, 52865.5363636364 * factor),
  (24 * 60 * 60, 49669.6409090909 * factor)
]

def getAvailability(t):
  dailyTime = t % (24 * 60 * 60)
  for (time, value) in points:
    if time > dailyTime:
      return oldValue + ( (value - oldValue) * (dailyTime - oldTime) / (time - oldTime) )
    oldValue = value
    oldTime = time



def main():
  devices = []
  shortOff = []
  nowOff = {}
  delayOn = []
  for j in xrange(0, numberOfDevices):
    rand = random.choice([1,2,3])
    if rand == 1:
      shortOff.append(ShortOff())
    elif rand == 2:
      delayOn.append(DelayOn())
    elif rand == 3:
      devices.append(Device())

  sys.stderr.write("Number␣of␣delayOn:␣"+str(len(delayOn))+"\n")
  sys.stderr.flush()

  constantConsumption = len(shortOff) * 1000 + len(devices) * 500
  sys.stderr.write("Constant␣consumption␣"+str(constantConsumption) + "\n")
  sys.stderr.flush()



  avg = -points[0][1]
  for (time, value) in points:
    avg += value
  avg /= len(points)-1

  alreadyScheduled = {}

  for t in xrange(0, 60 * 60 * 24 * 7, timestep):
    sys.stderr.write(str(t)+"\n")
    sys.stderr.flush()


    available = getAvailability(t)
    #noise = random.randint(-5 * 10 ** 5, 5 * 10 ** 5);
    #available += noise

    consumption = 0
    for d in devices:
      consumption += d.getConsumption(t)
    for d in shortOff:
      consumption += d.getConsumption(t)
    for d in delayOn:
      consumption += d.getConsumption(t)
    print t, available, consumption, available-consumption


    powerOnNumber = 0
    shutOffNumber = 0
    if consumption > available:
      delta = consumption - available
      shutOffNumber = int(delta / 1000)
    elif consumption < available:
      delta = available - consumption
      powerOnNumber = int(delta / 1000)

    #sys.stderr.write(str(shutOffNumber) + " " + str(powerOnNumber) + "\n")
    #sys.stderr.flush()


    i = 0
    while i < len(shortOff) and shutOffNumber > 0:
      if shortOff[i].shutOffPossible(t):
        shortOff[i].shutOff(t)
        shutOffNumber -= 1
        if not i in nowOff:
          nowOff[i] = True
      i += 1
    i = 0
    lenNowOff = len(nowOff)
    while i < lenNowOff and powerOnNumber > 0:
      (key, value) = nowOff.popitem()
      if not shortOff[key].isOn(t):
        shortOff[key].powerOn(t)
        powerOnNumber -= 1
      i += 1

    counter = 0
    for d in delayOn:
      if d.schedulable(t):
        counter += 1
        deadline = d.slotStart + d.slotLength - d.onTime
        startAt = deadline
        largest = -1000000000000000000
        for time in xrange(t, deadline, timestep):
          temp = getAvailability(time) - constantConsumption - d.consumption
          if time in alreadyScheduled:
            temp -= alreadyScheduled[time]

          if temp >= 0:
            startAt = time
            break;
          elif temp > largest:
            l = deadline - t
            if time - t > l / 2:
              factor = 1 - (time - t) / l
            else:
              factor = (time - t) / l
            #factor = ((deadline - t) - (time - t)) / (deadline - t)
            startAt = time - factor * d.onTime
            largest = temp

        d.powerOn(startAt)
        for duringOn in xrange(startAt, startAt + d.onTime, timestep):
          if duringOn in alreadyScheduled:
            alreadyScheduled[duringOn] += d.consumption
          else:
            alreadyScheduled[duringOn] = d.consumption

    sys.stderr.write(”Scheduling␣” +  str(counter) + ”␣devices\n”)
    sys.stderr.flush()


  for t in xrange(60 * 60 * 24 * 7, 2 * 60 * 60 * 24 * 7, timestep):
    available = getAvailability(t)
    consumption = 0
    for d in devices:
      consumption += d.getConsumption(t)
    for d in shortOff:
      consumption += d.getConsumption(t)
    for d in delayOn:
      consumption += d.getConsumption(t)
    print t, available, consumption, available-consumption

  nowOff = {}
  for t in xrange(2 * 60 * 60 * 24 * 7, 3 * 60 * 60 * 24 * 7, timestep):
    available = getAvailability(t)
    consumption = 0
    for d in devices:
      consumption += d.getConsumption(t)
    for d in shortOff:
      consumption += d.getConsumption(t)
    for d in delayOn:
      consumption += d.getConsumption(t)
    delta =  available-consumption
    print t, available, consumption, delta

    powerOnNumber = 0
    shutOffNumber = 0
    if consumption > available:
      delta = consumption - available
      shutOffNumber = int(delta / 1000)
    elif consumption < available:
      delta = available - consumption
      powerOnNumber = int(delta / 1000)


    i = 0
    while i < len(shortOff) and shutOffNumber > 0:
      if shortOff[i].shutOffPossible(t):
        shortOff[i].shutOff(t)
        shutOffNumber -= 1
        if not i in nowOff:
          nowOff[i] = True
      i += 1

    i = 0
    lenNowOff = len(nowOff)
    while i < lenNowOff and powerOnNumber > 0:
      (key, value) = nowOff.popitem()
      if not shortOff[key].isOn(t):
        shortOff[key].powerOn(t)
        powerOnNumber -= 1
      i += 1
    i = 0
    while i < len(delayOn) and powerOnNumber > 0:
      if delayOn[i].powerOnPossible(t):
        delayOn[i].powerOn(t)
        powerOnNumber -= 1
      i += 1











main()
```

Listing B.16: simulation.py

```
#!/usr/bin/python

import sys

f = open(sys.argv[1], "r")
integral = 0
for line in f:
  l = line.split()
  time = l[0]
  available = float(l[1])
  consumption = float(l[2])
  delta = abs(available - consumption)
  integral += delta * (6.0 / 360.0)
  print time, available, consumption, integral
```

Listing B.17: integrate.py