SEMINAR PAPER

openHAB – Empowering the Smart Home
History, Concepts, Examples

Author
Manuel Raffel
h0850648

Class Number	4289
Class Title	IS Projektseminar
Instructor	ao.Univ.Prof. Mag. Dr. Rony G. Flatscher

Abstract

This seminar paper handles the open source software openHAB, its history, concepts and provides examples for its use.
An historical overview on smart homes in general and openHAB in particular is given. The enabling technology is briefly explained and the core architecture of the software is depicted in great detail. Following this, an isolated example on how to use the system for a specific situation is discussed step by step.
The paper is concluded with a summary of the main messages of each section and recent developments and their impact on the future of openHAB are brought forward.

Contents

1 Introduction

While many may have heard of a "smart home" before, few will have a concrete understanding as of what exactly is meant with it. Neither space efficiency, building technique, usage of solar power nor waste water recycling does make a house smart. Although they often include these kind of things, what makes a smart home actually smart is the interactive technology it contains. [Harper 2003]

Albeit the level of interactivity varies between smart homes, they all have in common that they contain intelligent objects. There are different levels of communication of information within and beyond the smart home. Some homes can learn from the behaviour of their inhabitant and some even maintain constant awareness of occupants and objects. Starting from a certain level of interaction and abilities, all this needs to be controlled by a central and intelligent system. [Harper 2003]

OpenHAB (Open Home Automation Bus) is a project that acts as such a central system. Being open source, it is actively worked on by a large amount of people. Additionally, this means that everyone using it can adapt the code to his own needs, if necessary. The system wants to achieve a high level of compatibility and usability by providing a modular structure and a high number of supported technologies. [OpenHAB 2014]

The aim of this paper is to give an overview of the capabilities and concepts of openHAB as well as provide a brief outlook of things to come and latest developments. To achieve this, the following content is discussed:

- A brief overview on the history of smart homes in general and openHAB in particular (see Section 2).
- The enabling technology for this kind of connectivity in order to provide a deeper understanding for the system (see Section 3).
- The architecture behind the modular system of openHAB and its core components (see Section 4).
- An isolated example that leads through all steps necessary to set up the system introducing some of the features of openHAB (see Section 5).

The paper is then concluded by summarizing its content and providing an outlook for future development directions (see Section 6).

2 History

Although some hobbyists started modifying their houses in a "smart" direction as early as 1960, the term and practical application only just recently made its way to a greater number of people. According to Harper et. al., seedbed for the emergence of smart homes was the fast development of domestic technology from the introduction of electricity in the first quarter of the 20th century to the introduction of information technology in the last quarter. [Harper 2003]

Starting 1915, the first domestic machines requiring electricity were introduced into the households. Vacuum cleaners, food processors and sewing machines were advertised as time saving devices, though electricity was not yet very widespread. Until 1940, the proportion of households with access to electricity increased steadily, though most households only had it for lightning with none or just one socket for other devices. In the 1960s, when the ideal of the stay-at-home-housewive was overturned, and labour-saving devices thus became common in the home. In the 1980s and 90s the trend of increased technologisation continued. Colour television was a standard and the first multimedia PCs started to penetrate the domestic market, blurring the distinction between home and work. [Harper 2003]

Commercial interest in home automation lead to the foundation of a special interest group called "Smart House" by the National Association of Home Builders in the USA. Its aim was to push for the inclusion of necessary technology into the design of new homes. The concept finally became greater awareness through its appearance in life-style magazines such as Vanity Fair. Nevertheless, actual usage in private households remained low. [Harper 2003] Until now, the number has increased and is supposed to continue its rise. [Wikipedia 2014d]

OpenHAB was initially developed by Kai Kreuzer, a software architect with numerous years of experience. Initially, he was looking for an open source software to control the systems he was installing in his own house. He found but one, an old piece of software which was hard to extend and maintain. Therefore, in 2010 he decided to write his own. At the time of writing this paper, OpenHAB had reached version 1.5 and was used in over 1.000 installations. [InnoQ 2014]

3 Enabling Technology

OpenHAB is based on several different layers of technology that provide the needed abilities for the system to be interactive. This section explains them in a general way to establish the necessary foundation needed to understand openHAB itself.

3.1 Protocols

Home automation devices rely on a possibility to communicate with each other and the central server. Over the course of time, several technologies that can achieve this emerged. Some of them and their characteristics in terms of transmission type can be found in Table 1. Furthermore, the following sections will give an exemplary description of some.

Protocol	Wired[1]	Powerline[2]	Wireless[3]
C-Bus	yes	no	yes
EnOcean	no	no	yes
Insteon	no	yes	yes
KNX	yes	yes	yes
Universal Powerline Bus (UPB)	no	yes	no
X10	no	yes	yes
Zigbee	no	no	yes
Z-Wave	no	no	yes

Table 1: Smart Home Protocols [HomeAutomation 2014]

3.1.1 EnOcean

EnOcean is a purely wireless technology that is used primarily in automation systems, but also in industry, logistics and smart homes. It incorporates a proprietary wireless standard that is ratified as an international ISO standard. [EnOcean 2014]

1 Wired: A special cable has to be installed that connects all smart devices in order to allow communication between them [HomeAutomation 2014]. Typically, a special type of wiring to cancel out electromagnetic interference from external sources is used, a so-called twisted pair cabling [TwistedPair 2014].

2 Powerline: Transmission of data over a conductor that is also used for normal AC power transmission. This is achieved by adding a modulated carrier signal to the wiring system. Therefore, no additional wiring has to be installed as the existing AC cables can be used. [PowerLine 2014]

3 Wireless: Transmission of data between two ore more points that are not connected by an electrical conductor [Wireless 2014]. Smart home systems use a wide range of technologies and frequencies for their wireless connectivity [HomeAutomation 2014].

What makes this technology particularly interesting for use in home automation is its energy-harvesting capability. EnOcean is using a combination of electromagnetic, solar and thermoelectric energy converters to generate energy out of environmental energy fluctuations. Because of this, the use of e.g. wire- and battery-free light switches becomes possible. Other usages include light sensors, temperature sensors and even key card switches. [EnOcean 2014]

3.1.2 KNX

KNX is based on three previous home automation standards, the European Home Systems Protocol (EHS), BatiBUS and the European Installation Bus (EIB) and., like EnOcean, ratified as an international ISO standard. [KNX 2014a]

The protocol defines several physical communication media's including standard wiring, powerline networking, radio and also infrared as well as ethernet. It appears to be the most used standard in home automation. [KNX 2014a] As of 15th June 2014, the KNX Association had 352 members/manufacturers from 37 distinct countries [KNX 2014b]. Furthermore, it has partnership agreements with more than 30.000 installer companies in 100 countries and over 60 technical universities as well as more than 150 training centres. Its main benefit is that every of the more than 7.000 KNX certified products has been conformity tested by KNX accredited third party test labs. Therefore, components of different manufacturers are guaranteed to be compatible with each other. [KNX 2014a]

3.1.3 X10

X10 is probably the oldest of the nowadays used protocols. It primarily uses powerline but also defines a wireless radio based communication. Having been developed as early as 1975, it was the first general purpose home automation network technology. [X10 2014]

Although it is still widely used, the age of X10 means there are some shortcomings that newer protocols do not have. This includes commands getting lost if two signals are transmitted ad the same time through which they may collide. Also, the protocol is slow, as it takes roughly three quarters of a second to transmit a command. [X10 2014]

3.2 Open Service Gateway initiative (OSGi)

The Open Service Gateway initiative (OSGi) is a set of specifications that describe a modular component system for the Java programming language. It enables a development model where applications are dynamically composed of many different components. [OSGi 2011] Because of this, it is possible to add or remove functionality during runtime without stopping the application. Several well-known applications, including the Java IDE NetBeans and IBM's WebSphere Application Server, are based on OSGi. [OSGi 2014]

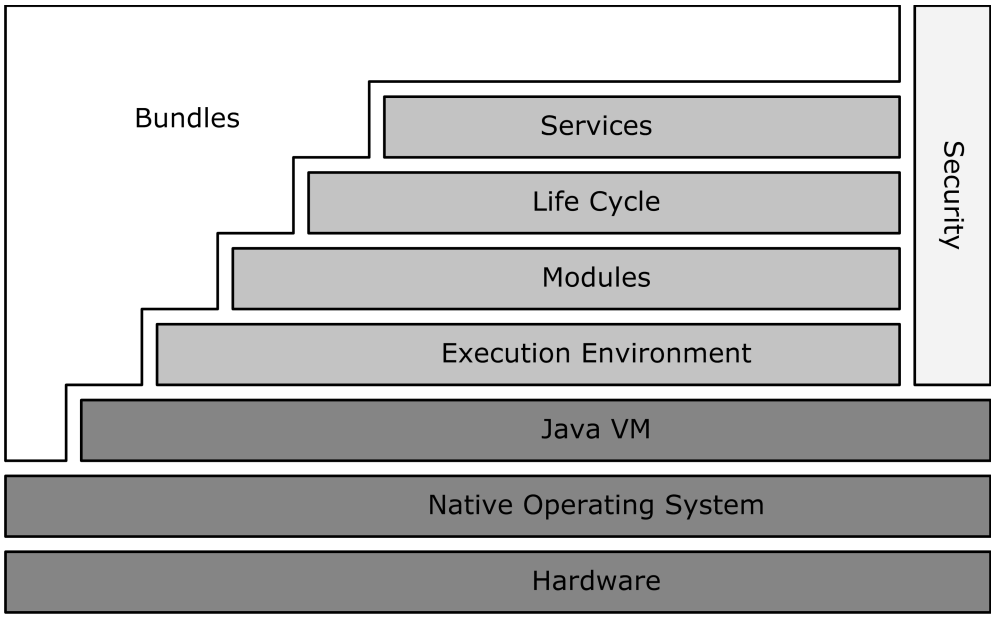

Figure 1: OSGi layers [OSGi 2011] [OSGi 2014]

Figure 1 shows the model of OSGi which includes the following layers: [OSGi 2011] [OSGi 2014]

- Bundles

  A Bundle is a piece of code written in Java that provides a certain functionality. It can be loaded into the OSGi framework where it interacts with all layers.

- Services

  The Service Layer maintains a registry of all loaded bundles and their functionality. Upon being loaded into OSGi, a bundle registers its interfaces in the service registry.

- Life Cycle

  This layer provides a life cycle API to bundles, i.e. functionality to start, stop, install, update and uninstall bundles.

- Modules
  Provides the modularization capabilities for OSGi. Incorporates strict rules for sharing Java packages between bundles or hiding packages from other bundles.
- Execution Environment
  Defines the methods and classes available in a specific execution environment, e.g. Java ME or the JRE.
- Security
  Handles overall security by limiting the functionality of bundles to pre-defined capabilities while also maintaining runtime interaction with the security layer integrated in Java.
- Java VM
  The Java Virtual Machine (VM) provides the platform-independent interface for OSGi to run on.
- Native Operation System / Hardware
  The Operating System (OS) that runs on the machine where OSGi is installed. As OSGi is based on Java, it can run on virtually any existing OS and hardware architecture.

3.3 Representational State Transfer (REST)

Representational State Transfer (REST) is a software architectural style to read, create, update or delete information on a server by using standard HTTP calls. It provides a simple alternative to other web service mechanisms like the Simple Object Access Protocol (SOAP) or Remote Procedure Call (RPC). Furthermore, REST is stateless, i.e. there is no information of the clients current state being stored on the server. This implies that each request has to contain all necessary information for the server to determine what he has to send back to the client but allows for an unproblematic way of exchanging server functionality as no state information can be lost. [REST 2014]

4 Architecture

While the openHAB runtime includes all the technologies explained in Section 3, its actual architecture is more complex as can be seen in Figure 2.

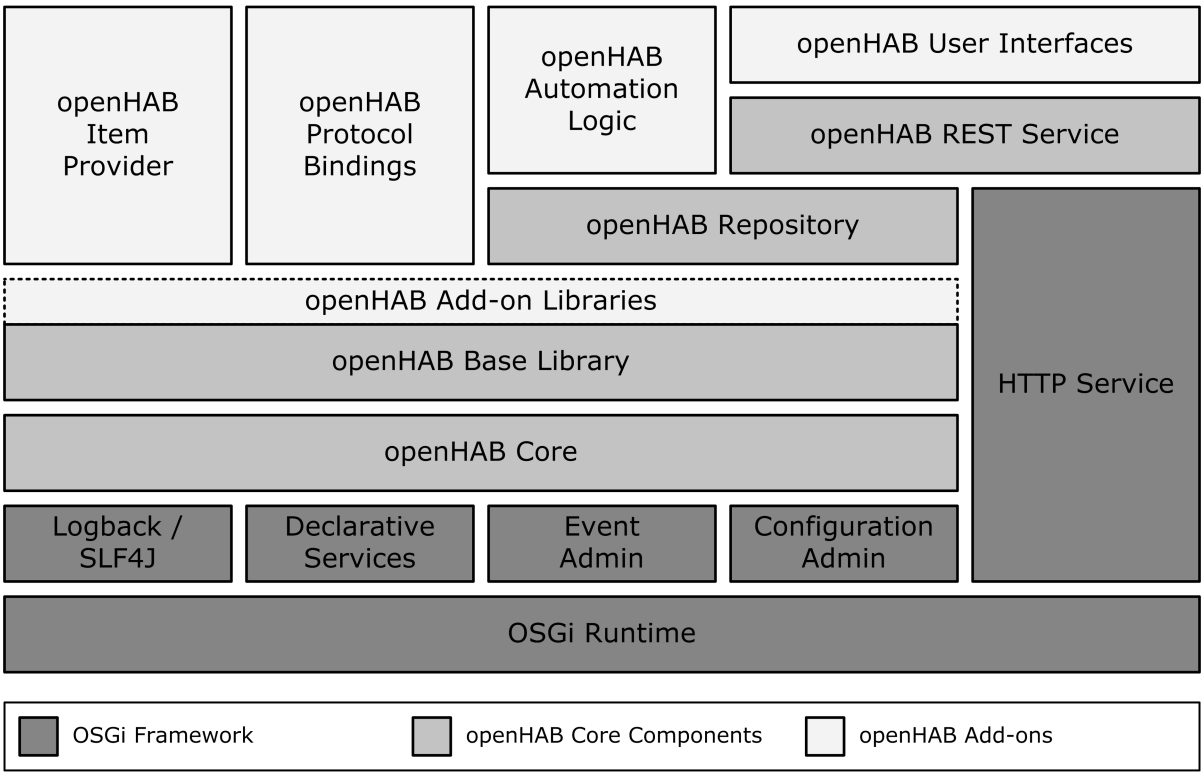

Figure 2: The openHAB Architecture. [OpenHAB 2014]

All units shown in Figure 2 are OSGi bundles or categories of bundles dedicated to specific purposes within the openHAB runtime [OpenHAB 2014]. A detailed explanation of them can be found in the following sections.

4.1 Core

The openHAB Core bundle constitutes the main foundation of the openHAB runtime. It contains basic definitions for all other bundles to build on and introduces two important concepts.

4.1.1 Items

An item is an abstract object that can be read from or written to. It normally refers to a particular device in the physical world or some value of it. E.g., it could be bound to a specific switch that is connected to openHAB via KNX. As the items and their values are also stored in a central item registry, the repository (see Section 4.3), the corresponding item will always refer to the current state of the switch (e.g. on or off). [OpenHAB 2014]

4.1.2 Event Bus

The most vital base service of the openHAB runtime, the event bus (see Figure 3), is also included in the core bundle.

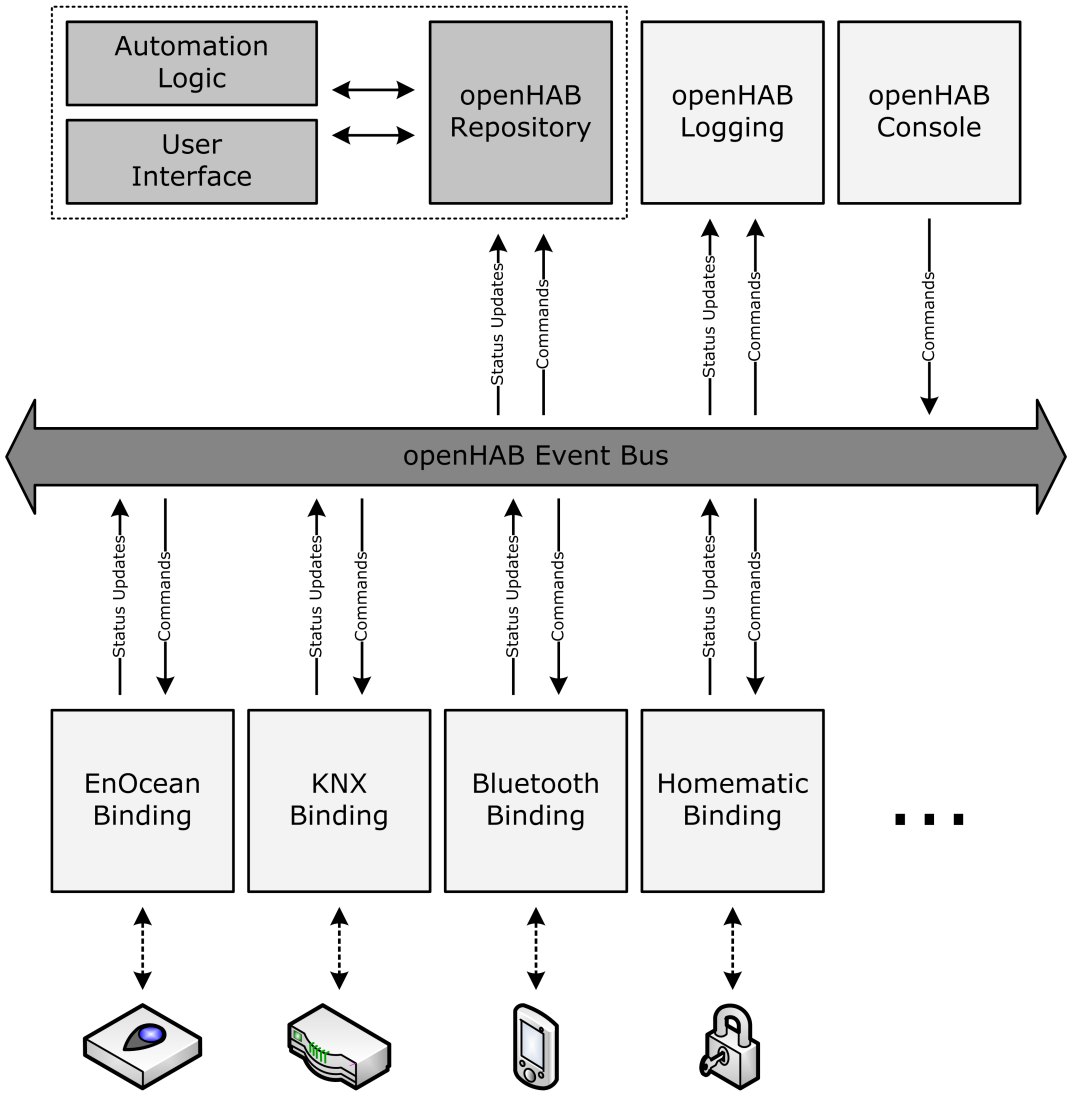

Figure 3: The openHAB Event Bus. [OpenHAB 2014]

The bus is used by all other bundles that do not require stateful behaviour, i.e. the server keeping track of the bundles status. Two kinds of events typically are transmitted over the bus:

- Commands, which set off an action or a state change for a particular item.
- Status updates, which inform about a state change of some item, e.g. as response to a command or when a movement sensor was triggered.

It is recommended that all protocol bindings, which provide the link to the physical devices, communicate via the event bus to make sure that the coupling between the bundles remains on a low level. [OpenHAB 2014]

4.2 Base Library

Library for all kind of features that are supposed to be available in every openHAB installation. At the time of writing of this paper, only a feature to integrate telephony into openHAB was in this bundle. [OpenHAB 2014]

4.3 Repository

As not all functionality can be covered purely by stateless services like the event bus, openHAB also offers a repository which keeps track of the current status of all items. It listens on the event bus and updates stored items according to status updates, as can be seen in Figure 3. [OpenHAB 2014]

Whenever the need arises to know the current state of an item, the repository can be used. It avoids that each bundle has to keep track of item states themselves for their internal use and therefore makes sure that the state is in sync for all bundles. Furthermore, it provides the possibility to save the states to the file system or the database, so that they are kept upon an eventual restart of the system. [OpenHAB 2014]

Other bundles, like the automation logic, are highly dependent on the repository, as it needs to always be informed about the current item states. User interfaces also depend on it, as the underlying REST service has to transmit up-to-date data to the client. [OpenHAB 2014]

4.4 REST Service

The REST service provides an API for a variety of purposes. Firstly, it can be used to access all items used in the system. A list of them can be retrieved including their current state or they can be accessed individually. This includes the possibility to send commands to an item. [OpenHAB 2014]

Secondly, all declared sitemaps are retrievable. A sitemap is a definition for the structure and content of different UI screens and is explained in more detail in Section 4.5. Of course, it is also possible to access the details of a specific sitemap. [OpenHAB 2014]

Lastly, the API makes it possible to subscribe a client to an item, a group of items, an item state and a single page of a sitemap. The service will automatically send an update to the subscribed client if the subscribed resource changes. [OpenHAB 2014]

4.5 User Interfaces

The user interface (UI) of openHAB is built with so-called sitemaps. A sitemap basically is a textual configuration file that consists of a tree structure of widgets and defines different pages of the UI and their content. A widget can be associated to an item to show the status and according control elements. [OpenHAB 2014]

4.6 Automation Logic

OpenHABs automation logic consists of two different components, scripts and rules. [OpenHAB 2014]

Scripts are blocks of code that can be defined by the user and stored at a particular location to be reusable from different places. They use openHABs own expression language which is similar to Java but includes several features that allow writing concise code. [OpenHAB 2014]

Rules are what makes the real home automation part in openHAB. A rule consists of one or more triggers, i.e. conditions that start the execution of a rule. Triggers can be item-based, time-based or system-based, thus either reacting on events on the event bus, on certain time constraints or on changes in the system status, e.g. shutdown. [OpenHAB 2014]

4.7 Protocol Bindings

Bindings are a major feature of openHAB, as they provide the necessary communication abilities for different kinds of underlying technology. With the help of a binding, openHAB can e.g. access serial devices or the KNX bus. As of 30th May 2014, openHAB provided 74 distinct bindings. A comprehensive list can be found in Appendix A, Table 6. [OpenHAB 2014]

4.8 Item Provider

This bundle extends the capabilities of the user interface and its sitemaps. An item provider can define what widget is to be used for a particular item. It can furthermore change its icon and label depending on the item's current state. This allows for a dynamically, i.e. at runtime, assembled user interface. The sitemap only defines the display of a certain group, while the actual content of the group is taken from the group's item provider. [OpenHAB 2014]

5 Practical Usage

This section provides a guide containing the necessary steps to set up openHAB and simulate a basic home automation scenario. It is intended to show the uncomplicatedness of the whole process. All following steps were conducted on a machine running a 64bit version of Windows 7 Ultimate SP1, but were afterwards replicated on a Mac running OS X version 10.9.3 and in a virtual machine running the Linux distribution Fedora 20 without any problem whatsoever. The used version of openHAB was initially 1.4.0, but after release of its latest version 1.5.0 on 15th June 2014 all steps were redone to make sure the functionality described here is consistent between both versions.

The setting for this guide is a house with one floor and several rooms that incorporate numerous devices linked together with a KNX bus. OpenHAB is the software of choice to run all aspects of home automation in this environment. As a first step, the corridor and the office are to be made accessible and controllable by an internet platform. The corridor comprises the main door of the house, which has a sensor to capture if its opened or closed as well as a light at the ceiling. The office too has a light at the ceiling as well as a light for the worktable. Additionally, the two windows in the room have sensors that register if they are opened or closed. Furthermore, one temperature sensor is installed on the outside of the windows and one in the room itself. The windows also have motorized roller shutters that are connected to the KNX bus. The setting will be referred to as DemoHouse in the remainder of this paper.

For later reference, a comprehensive collection of all files written in this section can be found in Appendix C.

5.1 Setup

The latest version of openHAB can be downloaded from its website, `http://www.openhab.org/`. The components available for download can be found directly at `http://www.openhab.org/downloads.html`. For a basic setup only the "openHAB Runtime Core" is necessary. In addition, several features required for DemoHouse depend on additional bundles. All bundles can be acquired at once by downloading the "Addons" package. As openHAB is written in Java, a Java Runtime Environment (JRE) is also necessary. An appropriate version can be downloaded at `http://www.java.com/`.

OpenHAB Core comes packaged in a ZIP archive without any installer whatsoever and therefore has to be copied manually into the desired directory which is not allowed to have any blank in its path. For reasons of simplicity `C:\openHAB\` was used in the course of this paper. After unpacking the archive, several files and directories can be found in the directory. For further reference, the most important ones needed to configure and operate openHAB are listed in alphabetical order in Table 2.

File / Directory	Description
openHAB	Folder that contains the main installation of openHAB.
├ addons	Folder for all bundles that extend functionality.
├ configurations	Folder that contains all configuration files.
│ ├ items	Folder that contains all item definitions.
│ ├ persistence	Folder that contains all configurations for persistence bindings.
│ ├ rules	Folder that contains all rule definitions.
│ ├ scripts	Folder that contains all scripts.
│ ├ sitemaps	Folder that contains all sitemaps.
│ ├ openhab_default.cfg	File that contains the default configuration for openHAB.
│ └ users.cfg	File that contains user information for authentication.
├ logs	Folder that contains log files.
│ └ openhab.log	File that contains the log outputted by openHAB.
├ start.bat	File that starts openHAB on a Windows based machine.
└ start.sh	File that starts openHAB on a Unix based machine.

Table 2: Important files and folders in the installation directory.

OpenHAB can now be started by executing the file `start.bat`. This starts a headless Java application, i.e. a standard console window with only text output. As of yet, no bindings whatsoever have

been included, neither has openHAB been configured. It is running nevertheless and shows the output depicted in Listing 1.

```
1  Launching the openHAB runtime...
2  osgi> 10:00:01.509 WARN  o.o.c.core.ConfigDispatcher[:172] - Main
   openHAB configuration file 'configurations/openhab.cfg
3  ' does not exist.
4  10:00:02.513 INFO  o.o.c.internal.CoreActivator[:61] - openHAB runtime
   has been started (v1.5.0).
5  10:00:03.018 INFO  o.o.i.s.i.DiscoveryServiceImpl[:72] - mDNS service
   has been started
6  10:00:04.493 INFO  o.o.io.rest.RESTApplication[:143] - Started REST API
   at /rest
7  10:00:05.154 INFO  o.o.u.w.i.s.WebAppServlet[:79] - Started Classic UI
   at /openhab.app
```

Listing 1: Initial output of the openHAB runtime.

To allow easier editing of all the configuration files, the openHAB Designer is a useful tool. It provides syntax highlighting for the language used and a navigation interface for all files. It too can be downloaded at `http://www.openhab.org/downloads.html` in the section "openHAB Designer".

It also comes packaged in a ZIP archive like the openHAB Core and has to be extracted to the desired destination directory. For reasons of simplicity `C:\openHAB Designer\` was used in the course of this paper. After unpacking the archive, the designer can be started by executing the file `openHAB-Designer.exe`. On first execution, the interface of the designer will show only empty tabs as the program does not know where to look for the openHAB configuration files. Therefore, it is necessary to click on the icon depicting a folder to the right of the tab labelled "Configurations". This will open a file browser where the path to the `configurations` folder within the openHAB directory has to be selected, in this case `C:\openHAB\configurations`. After confirming the selected location, the designer will show the content of this folder and is ready for use.

5.2 Configuration

To allow openHAB to start correctly, it needs to be configured. The configuration file has to be named `openhab.cfg` and be placed directly into the `configurations` subdirectory. As reference, the file `openhab_default.cfg` was already included in the archive. After copying and renaming it to `openhab.cfg` it is ready to be edited to particular needs.

A basic configuration is needed to run the system for DemoHouse, but it is short and requires only few options. The `folder:` values define the interval that openHAB waits between two scans for changes in its other configuration files. A default persistence provider has to be specified, this is more detailly explained Section 5.3.2. Finally, the refresh interval for the main configuration file itself has to be specified to allow changes being applied automatically when it changes. The content of the file can be seen in Listing 2.

```
1  folder:items=10,items
2  folder:sitemaps=10,sitemap
3  folder:rules=10,rules
4  folder:scripts=10,script
5  folder:persistence=10,persist
6
7  persistence:default=rrd4j
8
9  mainconfig:refresh=10
```

Listing 2: Content of the main configuration file for openHAB.

After the file has been saved, openHAB can be started again (see Listing 3 for console output). From this point on, there is no need to restart it when any configuration is changed or a bundle is added, as it will automatically reload the relevant information.

```
1  Launching the openHAB runtime...
2  osgi> 10:05:00.409 INFO  o.o.c.internal.CoreActivator[:61] - openHAB
   runtime has been started (v1.5.0).
3  10:05:01.929 INFO  o.o.i.s.i.DiscoveryServiceImpl[:72] - mDNS service
   has been started
4  10:05:02.380 INFO  o.o.io.rest.RESTApplication[:143] - Started REST API
   at /rest
5  10:05:03.335 INFO  o.o.u.w.i.s.WebAppServlet[:79] - Started Classic UI
   at /openhab.app
```

Listing 3: Output of the openHAB runtime after adding the main configuration file.

5.3 Adding Functionality

Now that openHAB itself is configured, the environment it is used for needs to be defined in several.

5.3.1 Creating Items

DemoHouse has several lights, shutters, sensors and according values that need to be defined as items in order to be able to use them in the system. This can be achieved by creating a file with the exten-

sion `.items` in the `openHAB/configurations/items` folder. The file for demo house was called `DemoHouse.items`. All items share the same syntax which is defined as shown in Listing 4.

```
1  itemtype itemname ["labeltext"] [<iconname>] [(group1,...)]
   [{bindingconfig}]
```

Listing 4: General syntax for defining items.

Parts in square brackets are thereby optional. The `itemtype` refers to one of the currently available types listed in Table 3.

Name	Description	Command Types
`Color`	Stores color information and allows commands for manipulation.	`ON`, `OFF`, `INCREASE`, `DECREASE`, `PERCENT`, `HSB`
`Contact`	Stores status of e.g. door or window contacts and allows commands for manipulation.	`OPEN`, `CLOSE`
`DateTime`	Stores date and time.	
`Dimmer`	Stores percentage values for dimmers and allows commands for manipulation.	`ON`, `OFF`, `INCREASE`, `DECREASE`, `PERCENT`
`Group`	Stores other items in it.	
`Number`	Stores arbitrary numbers.	
`Rollershutter`	Stores percentage values for blinds and allows commands for manipulation.	`UP`, `DOWN`, `STOP`, `MOVE`, `PERCENT`
`String`	Stores arbitrary text.	
`Switch`	Stores state of a switch and allows commands for manipulation.	`ON`, `OFF`

Table 3: The available item types. [OpenHAB 2014]

Groups are usually used to combine all items of one room together, so they can be visualized and even switched altogether. DemoHouse has some groups that are created as follows (see Listing 5):

```
1  Group g_all
2  Group g_shutters  (g_all)
3  Group g_contacts  (g_all)
4  Group g_temps     (g_all)
5
6  Group g_office    "Office"    <office>    (g_all)
7  Group g_corridor  "Corridor"  <corridor>  (g_all)
8
9  Group:Switch:OR(ON, OFF) g_lights  "All Lights [(%d)]"     (g_all)
```

Listing 5: Defining the groups for DemoHouse.

Line 1 of Listing 5 defines a group `g_all` for each and every item defined in DemoHouse. In line 2 to 4, a group for all shutters, contacts and temperatures is defined and included in `g_all`. Lines 6 and 7 define groups for all items in the office and in the corridor. Additionally, the icon to be used for this group in the visualization is defined by the term between the two angle brackets. OpenHAB looks for these icons in the folder `openHAB/webapps/images/`. Both groups are added to `g_all` as well.

Line 9 introduces an interesting concept of openHAB, the group summaries. Groups of a specific type, in this case indicated by the `:Switch` after `Group` can be summarized by a function. Possible functions are:

- `AVG` Displays the average of the items in the group, typically for items of type `Number` or with a percentage.
- `OR` Checks how many items in the group have a particular state.
- `AND` Checks if every item in the group has a particular state.
- `SUM` Displays the sum of the items in the group, typically for items of type `Number` or with a percentage.
- `MIN` Displays the minimum amount of the items in the group, typically for items of type `Number` or with a percentage.
- `MAX` Displays the maximum amount of the items in the group, typically for items of type `Number` or with a percentage.

The code in Listing 5 sets the group's label to "All Lights" followed by the amount of lights that are `ON` in the respective group.

Lights, which typically are either `ON` or `OFF`, can be integrated by adding items of type `Switch` into the configuration file. Dimmable lights can be integrated by using the type `Dimmer` which provides the additional functionality needed for this type of device. The lights installed in DemoHouse can be added as demonstrated in Listing 6.

Items for the windows roller shutters as well as window and door contacts need to be added likewise (see Listing 7).

```
1  Switch light_office_ceiling         "Ceiling"        (g_office,
   g_lights)
2  Switch light_office_table           "Table"          (g_office,
   g_lights)
3  Switch light_corridor_ceiling_front "Ceiling Front"  (g_corridor,
   g_lights)
4  Switch light_corridor_ceiling_back  "Ceiling Back"   (g_corridor,
   g_lights)
```

Listing 6: Defining the lights for DemoHouse.

```
1  Rollershutter shutter_office_left  "Office Left"    (g_office,
   g_shutters)
2  Rollershutter shutter_office_right "Office Right"   (g_office,
   g_shutters)
3
4  Contact window_office_left         "Office Left"    (g_office,
   g_contacts)
5  Contact window_office_right        "Office Right"   (g_office,
   g_contacts)
6  Contact door_corridor              "Door"           (g_corridor,
   g_contacts)
```

Listing 7: Defining roller shutters as well as window and door contacts for DemoHouse.

Finally, items to store the values of DemoHouse's temperature sensors have to be defined (see Listing 8). These items labels show "Temperature" followed by the stored temperature value with one decimal place.

```
1  Number temp_office_in   "Indoor [%.1f °C]"     <temperature> (g_office)
2  Number temp_office_out  "Outdoor [%.1f °C]"    <temperature> (g_office)
```

Listing 8: Defining items for sensor values in DemoHouse.

This concludes the definition of items for DemoHouse. All connect-able devices are now mapped in the system. Upon saving of the file, openHAB will automatically consider its content as shown in Listing 9.

```
1  10:10:00.587 INFO  o.o.m.c.i.ModelRepositoryImpl[:79] - Loading model
   'DemoHouse.items'
```

Listing 9: Output of the openHAB runtime after adding the items configuration.

5.3.2 Ensuring Persistence

To keep the current values of the items it is necessary to save them somewhere. Otherwise, in case of a system failure or any restart, the state of an item would not be known as long as it has not sent its

current status. As this usually happens only when e.g. a light switch is pressed, any visualization could not show the correct status of its items until then. For this purpose, a bundle that provides persistence has to be imported into the system. A number of possible solutions exist and are listed in Table 4.

Name	Description
db4o	Default persistence service for openHAB. Uses a database file and keeps all data that ever needed to be persistent, thus the file keeps growing as time goes by.
rrd4j	A round-robin database. Has a fixed allocated size which is used. If no space is left, the oldest item is overwritten with the newest one.
MySQL	Uses a MySQL[4] database that has to run on some database server.
MongoDB	A document-oriented and therefore highly performant database.
Sen.se	Uses the Open.Sen.se[5] website as storage location for persistence information.
Cosm	Uses the Xively[6] (former Cosm) website as storage location for persistence information.
Logging	Makes use of the OSGi integrated log mechanism to persist state information in ordinary log files.
Exec	Allows the execution of commands in the underlying operating system and can therefore be used very flexible with various programs.
InfluxDB	A time series database[7]. Has to connect to the database running on some database server.

Table 4: Available persistence bundles. [OpenHAB 2014]

For each persistence bundle that is used, a configuration file has to be created in the folder `openHAB/configurations/persistence`, named after the service and with the extension `.persist`. In it, the strategy for the persistence of specific items is defined. As there can be more persistence bundles, different modules can use different strategies, e.g. rrd4j for short-term and InfluxDB for long-term storage.

The file consists of two separate sections. The section `Strategies` defines when states are stored and has the following structure (see Listing 10):

4 MySQL: A open-source relational database management system (RDBMS). [MySQL 2014] All data is represented in terms of tuples, grouped into relations. [Relational 2014]
5 Open.Sen.se: Open platform for interconnecting devices, applications and humans. [OpenSenSe]
6 Xively: Cloud service similar to Open.Sen.se. [Xively 2014]
7 Time series database: A software system optimized for handling arrays of numbers indexed by time. [TimeSeries 2014]

```
1  Strategies
2  {
3        <strategyName1> : "<cronExpression1>"
4        <strategyNameX> : "<cronExpressionX>"
5        ...
6
7        default = <strategyNameX>, <strategyNameY>
8  }
```

Listing 10: General structure of the strategies section for persistence.

In this block, `<strategyNameX>` is an arbitrary name that refers to the according `<cronExpressionX>`. The expression is a string of six to seven sub-expressions separated by a white-space and defines when the strategy should be executed, e.g. `"0 0 * * * ?"` for every hour or `"0 0 0 * * ?"` for every day. A detailed description of this syntax can be found at the website of the underlying technology [CronTrigger 2014]. Furthermore, the following strategies are defined statically by the openHAB runtime:

- `everyChange` Stores the current value whenever it has changed.
- `everyUpdate` Stores the current value whenever it is updated, even if the value itself did not change.
- `restoreOnStartup` Restores the last saved value if it is undefined on startup.

Line 7 defines the default strategy to be employed on any item that has no strategy defined. Strategies for items are defined in the `Items` section that follows the `Strategies` section in the configuration file (see Listing 11).

```
1  Items
2  {
3        <itemList1> [-> "<alias1>"] : [strategy = <strategyName1>,
         <strategyName2>,...]
4        <itemListX> [-> "<aliasX>"] : [strategy = <strategyNameX>,
         <strategyNameY>,...]
5        ...
6  }
```

Listing 11: General structure of the items section for persistence.

Possible values for `<itemListX>` are thereby:

- `*` Defined strategies apply to all items in the system.

- `<itemName>` Defined strategies apply to the item identified by the given name. If the item is a group, the group value will be stored but not the values of its individual members.
- `<groupName>*` Defined strategies apply to all items in the group identified by the given name, but not to the group itself.

For DemoHouse, a combination of db4o and rrd4j has been chosen. The first one is intended for long-time storage of temperature sensor values so a chart can be shown on the data. Therefore, the values of all items in `g_temps` is persisted every hour. The latter is for up-to-date storage of every value in the two rooms. They are saved on every change and additionally every minute. Additionally, the values will be restored upon system restart. The final configuration files can be seen in Listing 12 and 13.

```
1  Strategies
2  {
3        everyHour : "0 0 * * * ?"
4  }
5
6  Items
7  {
8        g_temps* : strategy = everyHour
9  }
```

Listing 12: Persistence configuration using db4o.

```
1  Strategies
2  {
3        everyMinute : "0 * * * * ?"
4  }
5
6  Items
7  {
8        g_office*, g_corridor* : strategy = everyChange, everyMinute,
         restoreOnStartup
9  }
```

Listing 13: Persistence configuration using rrd4j.

OpenHAB will automatically consider the content of both files after they have been saved as shown in Listing 14.

```
1  10:15:00.374 INFO  o.o.m.c.i.ModelRepositoryImpl[:79] - Loading model
   'db4o.persist'
2  10:15:01.383 INFO  o.o.m.c.i.ModelRepositoryImpl[:79] - Loading model
   'rrd4j.persist'
```

Listing 14: Output of the openHAB runtime after adding the persistence configurations.

To finally enable both modules, the according bundles have to be copied from the "Addons" package downloaded in Section 5.1 to the `openHAB/addons/` folder. The files necessary are named `org.openhab.persistence.db4o-1.5.0.jar` for the db4o bundle and `org.openhab.persistence.rrd4j-1.5.0.jar` for the rd4j bundle.

5.3.3 Defining Rules

After having defined all the data that is needed to replicate the physical devices in openHAB, functionality can be added. In a realistic setting, this involves for example defining which light switch triggers what or at what measured wind strength the shutters will be brought down automatically. For DemoHouse, rules are mainly used to simulate a realistic setting. The rules have to be placed in files with the extension `.rules` and are expected to be located in the folder `openHAB/configurations/rules`. Variables defined within a rule file are only visible inside that one file and cannot be shared with rules from another file.

Listing 15 shows the code that initializes all items to random default values. The lights are thereby set to be `ON` or `OFF`, contacts to `OPENED` or `CLOSED`, roller shutters to some random position and the temperatures receive a random value somewhere around 25 degrees Celsius. All rules are defined to be executed on system startup, with the rule that sets the temperature sensors being additionally executed every minute to simulate environmental behaviour.

The configuration for the rules will automatically be considered by openHAB after saving the file (see Listing 16).

```
rule "Initialize simulated light states"
      when
            System started
      then
            g_lights?.members.forEach(
                  light | light.postUpdate(if(Math::random > 0.5)
            ON else OFF)
            )
end

rule "Initialize simulated contact states"
      when
            System started
      then
            g_contacts?.members.forEach(
                  contact | contact.postUpdate(if(Math::random >
            0.5) OPEN else CLOSED)
            )
end

rule "Initialize simulated roller shutter states"
      when
            System started
      then
            g_shutters?.members.forEach(
                  shutter | shutter.postUpdate((Math::random *
            10).intValue * 10)
            )
end

rule "Initialize simulated temperature sensors"
      when
            System started or
            Time cron "0 0/1 * * * ?"
      then
            g_temps?.members.forEach(
                  temperature | temperature.postUpdate(25.0 + (25.0
            - (Math::random * 50.0).intValue) / 10.0)
            )
end
```

Listing 15: Rules defined for DemoHouse.

```
10:20:00.374 INFO  o.o.m.c.i.ModelRepositoryImpl[:79] - Loading model
'db4o.persist'
10:20:01.383 INFO  o.o.m.c.i.ModelRepositoryImpl[:79] - Loading model
'rrd4j.persist'
```

Listing 16: Output of the openHAB runtime after adding the rules configuration.

5.3.4 Using Scripts

Scripts provide the possibility to have reusable pieces of code that can be used in rules and, with the proper binding, could even be called from within an event in a Google calender, therefore being executed at a predefined time. Each file with the ending `.script` is handled as distinct script identified by its file name. The file has to be placed within `openHAB/configuration/scripts` to be found by openHAB.

For DemoHouse a relatively simple script for "coming home" was written that, if activated, turns on all lights and opens the roller shutters. The file was named `ComingHome.script` and can therefore be called from within any rule through `callScript("ComingHome")`. The code of this script can be seen in Listing 17.

```
1  g_lights?.members.forEach(
2        light | light.postUpdate(ON)
3  )
4
5  g_shutters?.members.forEach(
6        shutter | shutter.postUpdate(UP)
7  )
```

Listing 17: The ComingHome script for DemoHouse.

The available scripts will be updated by openHAB after saving the file (see Listing 18).

```
1  10:25:00.985 INFO  o.o.m.c.i.ModelRepositoryImpl[:79] - Loading model
   'ComingHome.script'
```

Listing 18: Output of the openHAB runtime after adding a script.

5.4 Bindings

After setting up the basic functionality of DemoHouse for the first two rooms, openHAB needs to be connected to the physical world.

5.4.1 KNX

In this case, a KNX bus is installed and physically connects the devices to the system. To enable openHAB to connect itself to the bus, the KNX binding is needed. The according bundle has to be

copied from the "Addons" package downloaded in Section 5.1 to the `openHAB/addons/` folder. The file necessary is named `org.openhab.binding.knx-1.5.0.jar`.

Two steps are required to set up the bus connection. First, the settings needed to connect to the bus have to be added to the main configuration file `openhab.cfg`. As DemoHouse does not have an actual bus, no options have to be configured.

Second, the items created in Section 5.3.1 have to be adjusted to tell the system how it can access them. This is done by adding binding information to the `DemoHouse.items` file. Syntax for this information is shown in Listing 19.

```
1  knx="[<][<dptId>:]<mainGA>[[+[<]<listeningGA>]+[<]<listeningGA>...], [<]
   [<dptId>:]<mainGA>[[+[<]<listeningGA>]+[<]<listeningGA>...]"
```

Listing 19: General syntax for defining KNX information.

Parts in square brackets are thereby optional. Each of the parts of the string separated by a comma correspond to a KNX datapoint. This tells the system, which command has to be sent to which KNX address. E.g., the switch item in DemoHouse for the light above the office table supports the commands `ON` and `OFF`. It therefore needs to be configured to know where to send this commands to. It furthermore needs to know, where it is supposed to listen for status changes of this item. This is particularly important for functions that can be controlled with physical devices, like an actual light switch. If it is pressed, the KNX bus sends out a signal to a specific address. This signal has to be caught by openHAB to pass it on to the actual light and persist its changed state. Listing 20 shows how the item would look like with configured KNX binding.

```
1  Switch light_office_table "Table" (g_office, g_lights)
   { knx="1/2/2+0/2/2" }
```

Listing 20: Adding KNX information to DemoHouse's items.

The code above tells openHAB to send commands issued to this item to the KNX address 1/2/2, where it is executed. Normally, some kind of relay, which is connected to the bus, will listen for commands issued to this address and activate or deactivate current flow to the light. It further tells openHAB to listen for commands sent to the address 0/2/2, for example by a light switch connected to the bus, and relay the information to 1/2/2.

5.4.2 Bluetooth

The server on which openHAB is running on in DemoHouse is equipped with a Bluetooth module that allows connection with mobile phones. This shall be used to automatically trigger the "coming home" script implemented in Section 5.3.4 whenever the phone comes within reach of the server. To enable openHAB to use the Bluetooth module, the Bluetooth binding is needed. The according bundle has to be copied from the "Addons" package downloaded in Section 5.1 to the `openHAB/addons/` folder. The file necessary is named `org.openhab.binding.bluetooth-1.5.0.jar`. A successful integration into the running openHAB environment can be confirmed when the line shown in Listing 21 is displayed in the console.

```
1  BlueCove version 2.1.1-SNAPSHOT on winsock
```

Listing 21: Output of the openHAB runtime after successful loading of the Bluetooth bundle.

Apart from adjusting items and rules, no further configuration is needed for the bundle to work. To access data about devices in range, an additional item has to be created as shown in Listing 22.

```
1  Switch bluetooth_device_in_range   { bluetooth="54E43A6CA061" }
```

Listing 22: Adding an item to access Bluetooth information.

The switch created above will listen on the Bluetooth module and change its state to `ON` once the device with the Bluetooth address `"54E43A6CA061"` comes in range. It will change to `OFF` again once the device leaves the area. To trigger the "coming home" script a rule has to be added (see Listing 23).

```
1  rule "Call 'ComingHome' upon bluetooth connection"
2        when
3             Item bluetooth_device_in_range changed to ON
4        then
5             callScript("ComingHome")
6  end
```

Listing 23: Adding a rule to trigger the "coming home" script.

Upon saving of all edited files, the openHAB environment will reprocess the information within and start listening for Bluetooth devices. Listing 24 shows what is happening once the specified device comes within range.

```
1  10:30:00.548 INFO  runtime.busevents[:26] - bluetooth_device_in_range
   state updated to ON
2  10:30:00.551 INFO  runtime.busevents[:26] - light_office_ceiling state
   updated to ON
3  10:30:00.551 INFO  runtime.busevents[:26] - light_office_table state
   updated to ON
4  10:30:00.551 INFO  runtime.busevents[:26] - light_corridor_ceiling_front
   state updated to ON
5  10:30:00.552 INFO  runtime.busevents[:26] - light_corridor_ceiling_back
   state updated to ON
6  10:30:00.564 INFO  runtime.busevents[:26] - shutter_office_left state
   updated to UP
7  10:30:00.570 INFO  runtime.busevents[:26] - shutter_office_right state
   updated to UP
```

Listing 24: Output of the openHAB runtime upon detecting a specific Bluetooth device.

5.5 Visualization

To add an intuitive overview of all the status changes in DemoHouse, openHAB provides a built-in visualization system, the sitemaps. They are used to create elements of user interfaces to make the system accessible to various frontends. The sitemaps have to be placed in files with the extension `.sitemap` and are expected to be located in the folder `openHAB/configurations/sitemaps`. The syntax of a sitemap requires all elements to be surrounded by the `sitemap` block as shown in Listing 25.

```
1  sitemap DemoHouse label="DemoHouse"
2  {
3        ...
4  }
```

Listing 25: General syntax of a sitemap.

Upon saving of this file, the openHAB runtime automatically loads its content and makes it accessible as can be seen in Listing 26.

```
1  10:35:00.956 INFO  o.o.m.c.i.ModelRepositoryImpl[:79] - Loading model
   'default.sitemap'
```

Listing 26: Output of the openHAB runtime after adding a script.

Naming the sitemap `"default.sitemap"` allows to access the sitemap by pointing any browser to `http://127.0.0.1:8080/openhab.app` without any additional argument. Sitemaps with a different name can be accessed by appending the `parameter ?sitemap=NameOfSitemapFile` to the address. These few lines of code already provide a visualization with the configured label as title, as can be

seen in Appendix B, Figure 4. To add more content, other elements have to be added. Table 5 shows the numerous predefined elements that are available to be used in a sitemap definition file and provides a short description for each.

Name	Description
Colorpicker	Allows to pick a color, e.g. for a multicolour light bulb.
Chart	Renders a chart based on specified information.
Frame	Creates a visually separated area of items.
Group	Provides a page with all items of the specified group.
Image	Renders an image.
Switch	Renders a switch item
Selection	Renders a selection item
Slider	Renders a slider
Text	Renders a text element
Video	Displays a video.

Table 5: The available sitemap elements. [OpenHAB 2014]

To visualize some actual information from DemoHouse, a group element can be used as shown in Listing 27. A group automatically enables access to its contained items in a separate page. Figure 5 and 6, which can be found in Appendix B, show the current main page and the page when "Office" is clicked.

```
1  Frame
2  {
3        Group item=g_corridor label="Corridor" icon="corridor"
4        Group item=g_office label="Office" icon="office"
5  }
```

Listing 27: A frame containing group items to show some information in a sitemap.

Individual information, like the outside temperature, can be added as well. Furthermore, it is possible to add a chart which depicts the assigned items progression in a specified period, if persisted somewhere. The code to add this information can be seen in Listing 28 and the according visualizations are depicted in Figures 7 and 8 in Appendix B.

This concludes the visualization for DemoHouse. Overall, the built-in visualization capabilities of openHAB are legion. In addition, it is possible to implement a completely independent visualization by using the REST API of openHAB. This allows for even greater flexibility and customization needs.

```
1  Frame label="Temperature"  icon="temperature"
2  {
3        Text item=temp_office_out
4        {
5              Frame
6              {
7                    Chart item=temp_office_out period=3D refresh=600
   service="db4o"
8              }
9        }
10 }
```

Listing 28: A frame containing a text item and a chart to show individual information in a sitemap.

6 Summary and Outlook

As history has shown, pace of technological development has increased dramatically within the last 100 years. While initial propagation of electrical devices was fairly slow, nowadays new technologies are quickly adapted to. Given the increase in sales for smart home technology, it will be employed in numerous homes new and old in the not to distant future.

OpenHAB has found a solid base of contributors who build on its highly modular structure. With over 1000 installations, it is not a small project but has yet to find its way into mainstream awareness. Its architecture in combination with the publicly available source code make it the ideal platform for enthusiasts to build on, but it is lacking the comfortableness for standard users in terms of setup and configuration.

Nevertheless, it has countless practical applications. With a rather minimal effort to set it up, the focus can be put on a comprehensive configuration. Due to the large integrated functionality and huge amount of available extensions, it can be fitted to almost any situation. The incorporated visualization capabilities that can be accessed by any device eases initial effort for testing the functionality of the system. For a longer-term usage, a more professional look and feel would be desirable.

On 16th of June 2014, shortly before this paper was finished, Kai Kreuzer, the initiator of the project, announced the start of development for version 2.0 of openHAB. The core concept was contributed to the Eclipse Foundation and became the new Eclipse SmartHome project, a basic framework to build smart home solutions on top. The SmartHome project will be the foundation for version 2.0, which is going to put development focus on user comfort. This will pave the way for openHAB to become a viable alternative not only for computer enthusiasts but also for the standard user.

Appendix A – Tables

Name		
Asterisk	Ir-Trans	Pioneer-AVR
Astro	KNX	Plugwise
Bluetooth	Koubachi	PLCBus
Comfo Air	Leviton/HAI Omnilink	Pulseaudio
CUL	MAX!Cube	RFXCOM
CUPS	MiLight	Samsung TV
digitalSTROM	Modbus TCP	Serial
DMX512	MPD	Snmp
EnOcean	MQTT	Squeezebox
Epson Projector	MQTTitude	System Info
Exec	Neohub	Somfy URTSI II
Freebox	Netatmo	Sonos
Freeswitch	Network Health	Swegon
Fritz!Box	Nibe Heatpump	TCP/UDP
Fritz AHA	Nikobus	Tellstick
GPIO	Novelan/Luxtronic	TinkerForge
HAI/Leviton OmniLink	NTP	VDR
HDAnywhere	One-Wire	Velleman
Heatmiser	Onkyo AV Receiver	Wake-on-LAN
Homematic	Open Energy Monitor	Waterkotte EcoTouch
HTTP	OpenPaths	Withings
IEC 62056-21	OpenSprinkler	XBMC
IHC / ELKO	OSGi Configuration Admin	xPL
Insteon Hub	Philips Hue	Z-Wave
Insteon PLM	Piface	

Table 6: Available bindings for openHAB as of 30th May 2014. [OpenHAB 2014]

Appendix B – Figures

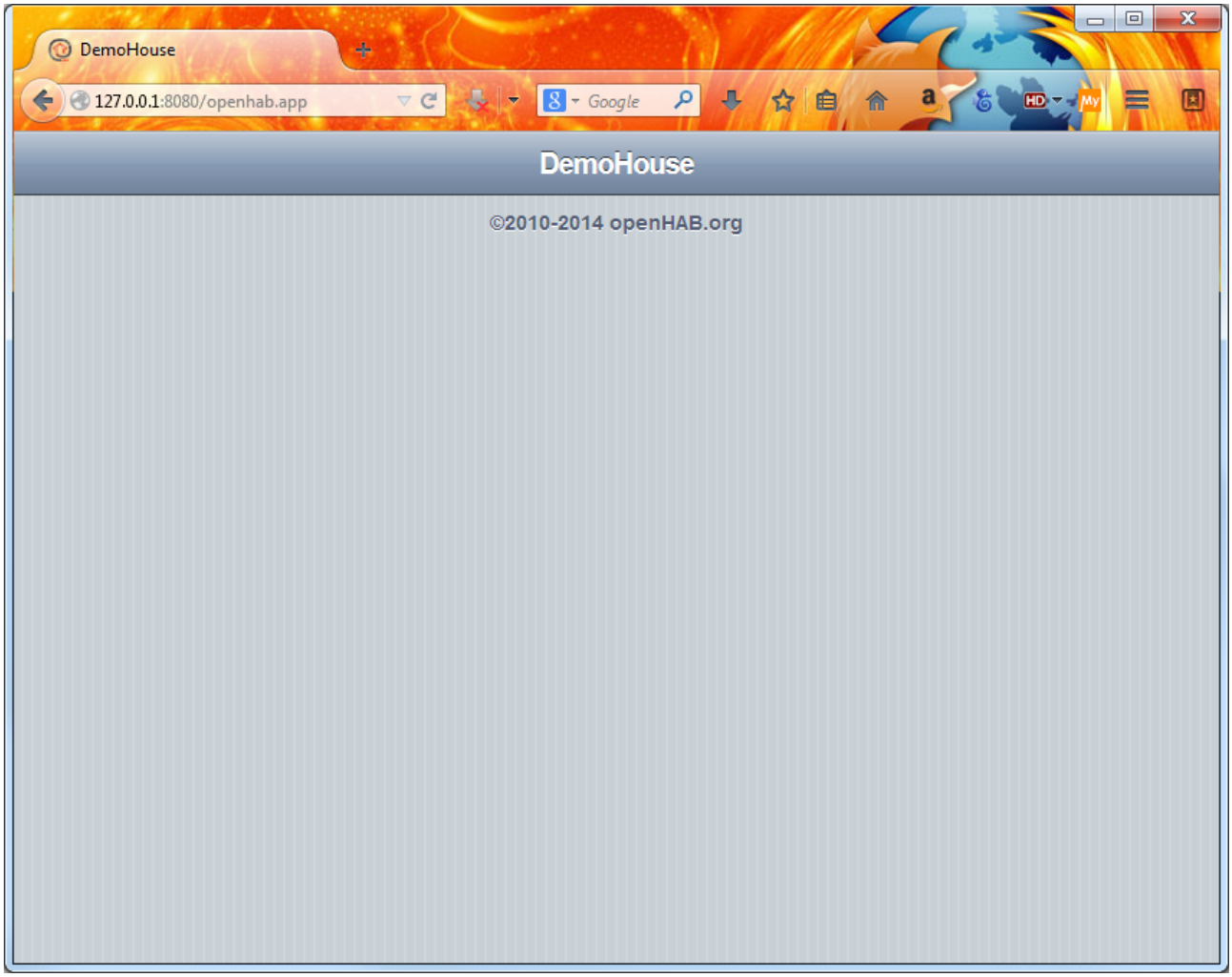

Figure 4: Visualization provided by the sitemap element.

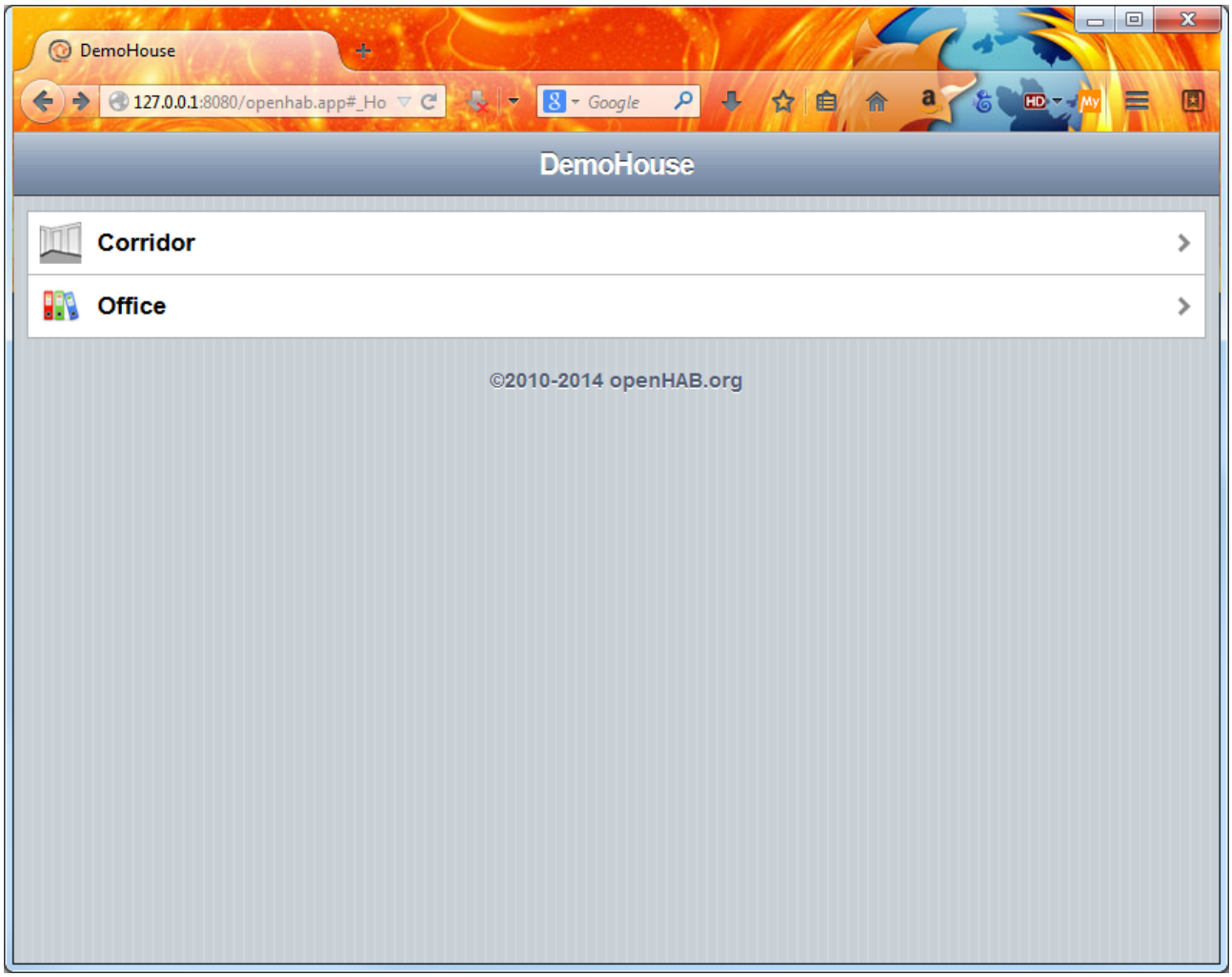

Figure 5: Visualization of item groups.

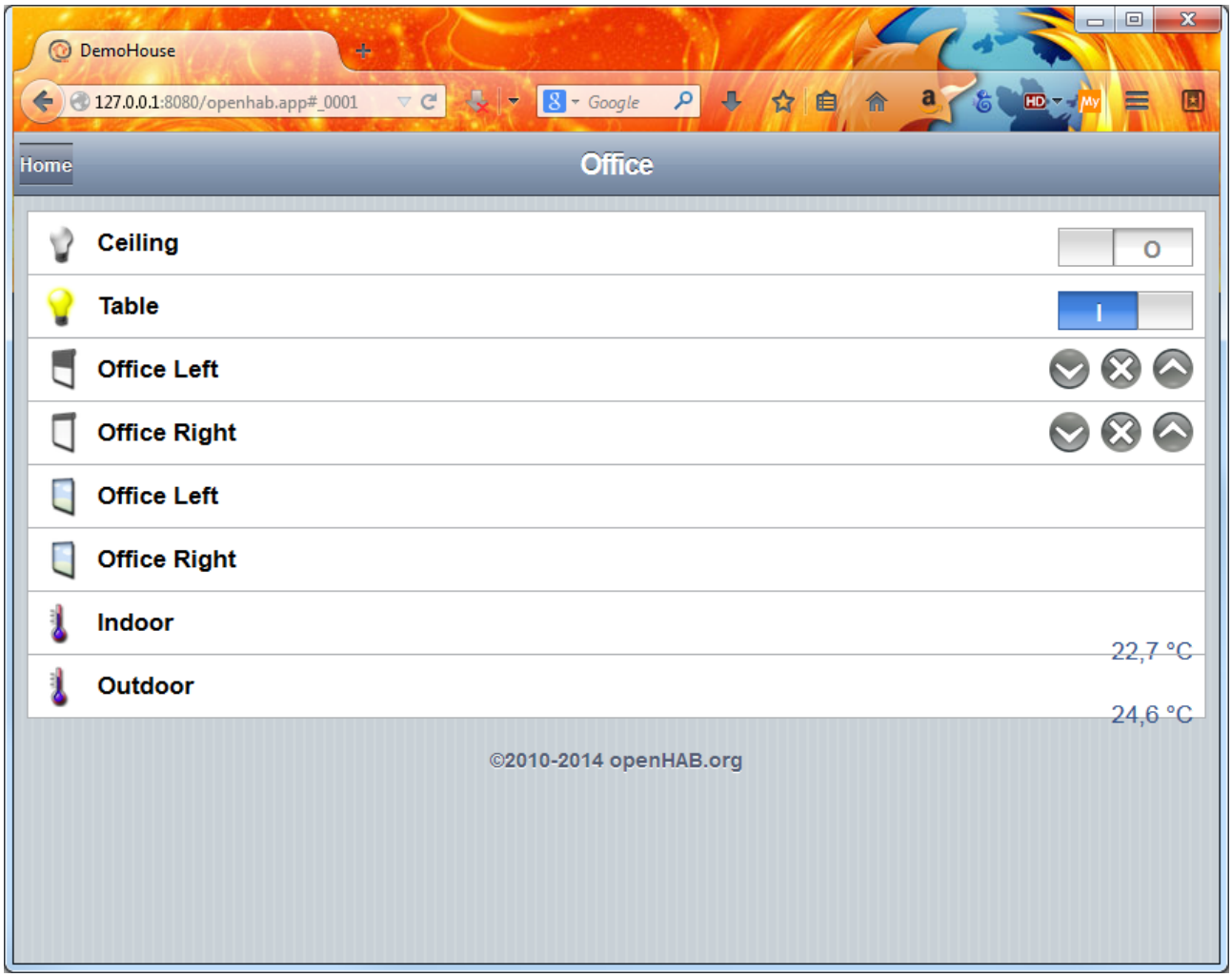

Figure 6: Visualization of items in item groups.

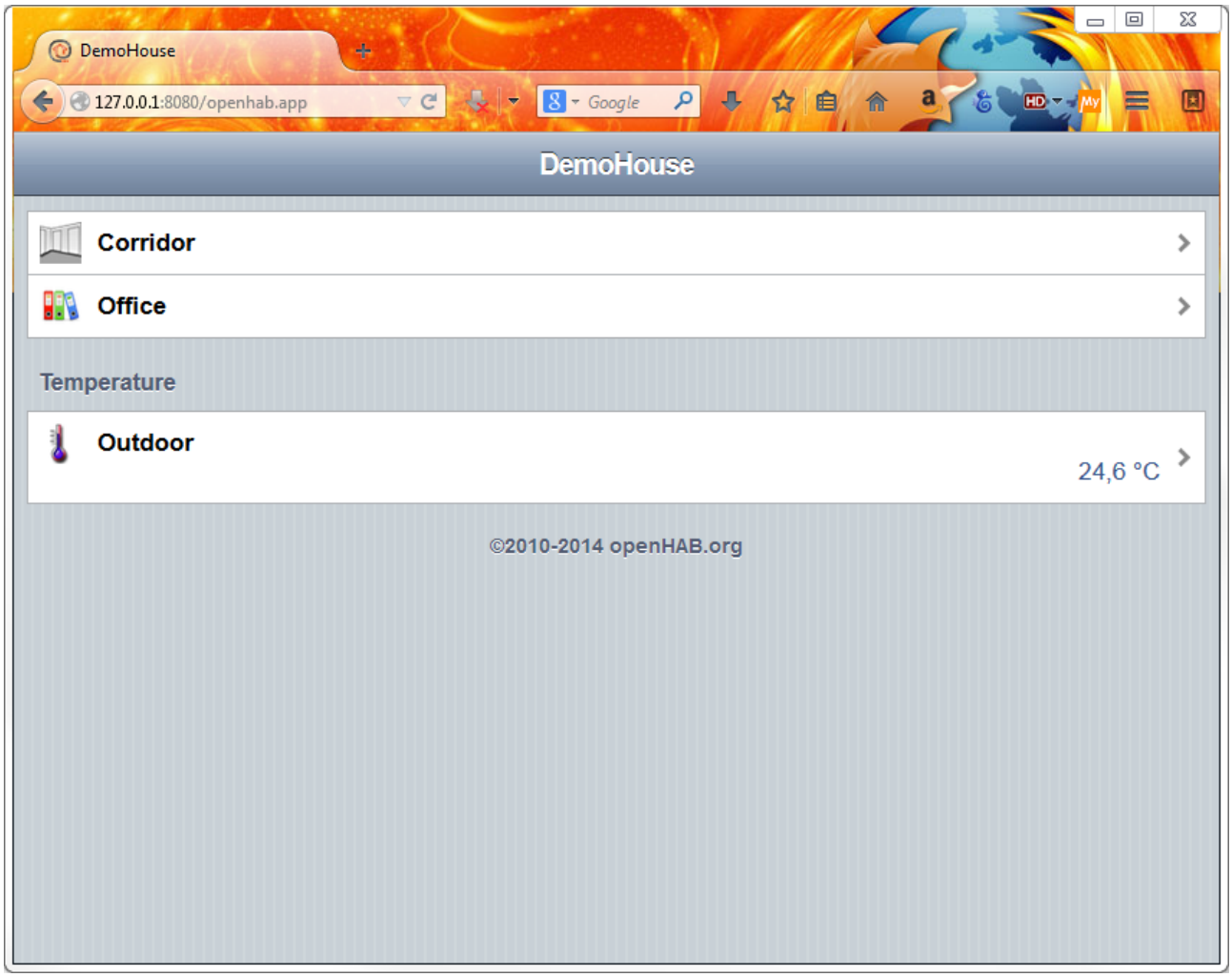

Figure 7: Visualization of individual values as text item.

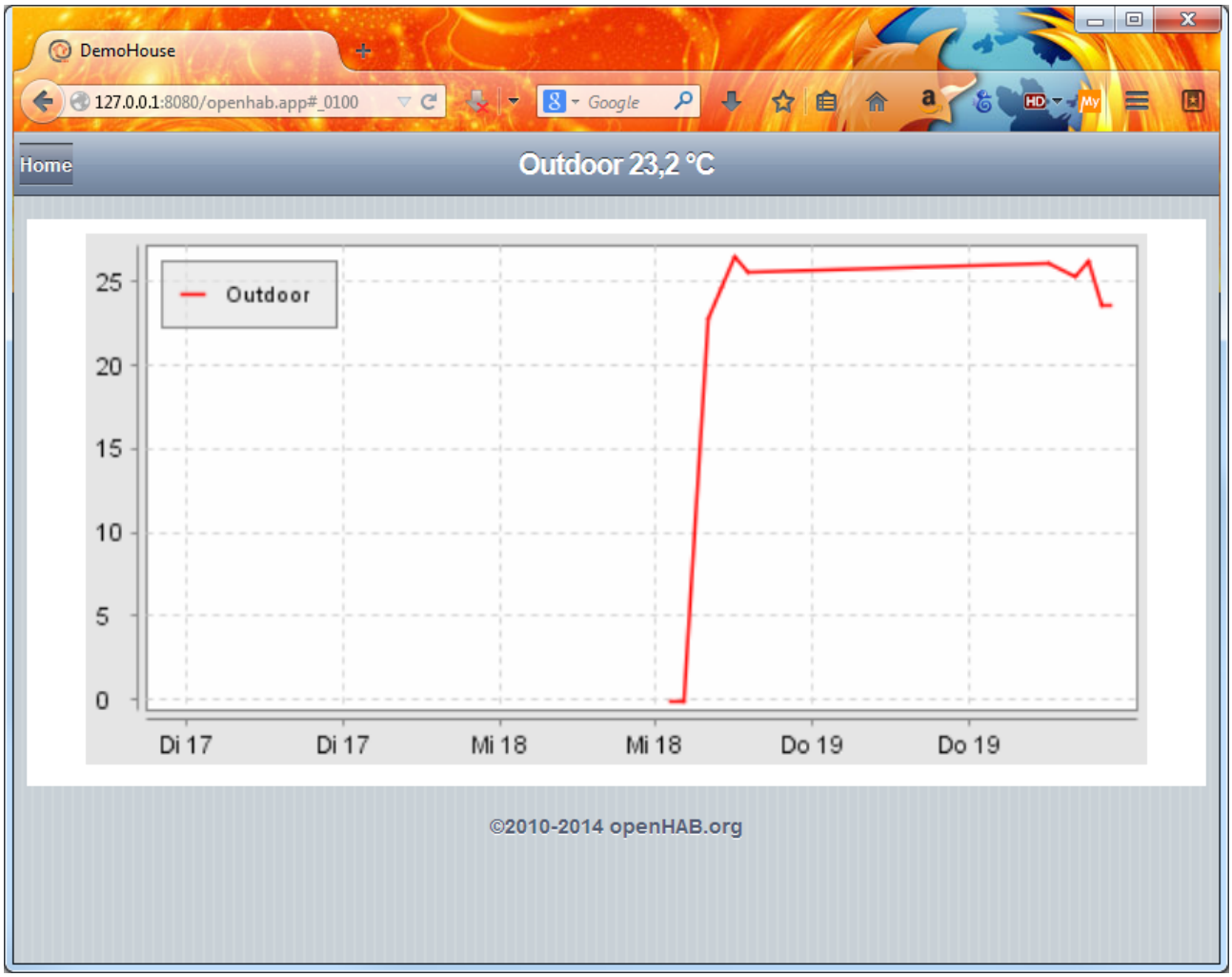

Figure 8: Visualization of historical values of an item in a dynamically generated graph.

Appendix C – Listings

```
folder:items=10,items
folder:sitemaps=10,sitemap
folder:rules=10,rules
folder:scripts=10,script
folder:persistence=10,persist

persistence:default=rrd4j

mainconfig:refresh=10
```

Listing 29: Consolidated version of openhab.cfg.

```
Strategies
{
	everyHour   : "0 0 * * * ?"
}

Items
{
	g_temps* : strategy = everyHour
}
```

Listing 30: Consolidated version of db4o.persist.

```
Strategies
{
	everyMinute : "0 * * * * ?"
}

Items
{
	g_office*, g_corridor* : strategy = everyChange, everyMinute,
restoreOnStartup
}
```

Listing 31: Consolidated version of rrd4j.persist.

```
g_lights?.members.forEach(
	light | light.postUpdate(ON)
)

g_shutters?.members.forEach(
	shutter | shutter.postUpdate(UP)
)
```

Listing 32: Consolidated version of ComingHome.script.

```
Group g_all
Group g_shutters  (g_all)
Group g_contacts  (g_all)
Group g_temps     (g_all)

Group g_office    "Office"    <office>    (g_all)
Group g_corridor  "Corridor"  <corridor>  (g_all)

Group:Switch:OR(ON, OFF)      g_lights    "All Lights [(%d)]"     (g_all)

Switch light_office_ceiling               "Ceiling"         (g_office,
g_lights)
Switch light_office_table                 "Table"           (g_office,
g_lights)   { knx="1/2/2+0/2/2" }
Switch light_corridor_ceiling_front       "Ceiling Front"   (g_corridor,
g_lights)
Switch light_corridor_ceiling_back        "Ceiling Back"    (g_corridor,
g_lights)

Rollershutter shutter_office_left         "Office Left"     (g_office,
g_shutters)
Rollershutter shutter_office_right        "Office Right"    (g_office,
g_shutters)

Contact window_office_left                "Office Left"     (g_office,
g_contacts)
Contact window_office_right               "Office Right"    (g_office,
g_contacts)
Contact door_corridor                     "Door"            (g_corridor,
g_contacts)

Number temp_office_in                     "Indoor [%.1f °C]"
      <temperature>     (g_office, g_temps)
Number temp_office_out                    "Outdoor [%.1f °C]"
      <temperature>     (g_office, g_temps)

Switch bluetooth_device_in_range    { bluetooth="54E43A6CA061" }
```

Listing 33: Consolidated version of DemoHouse.items.

```
rule "Initialize simulated light states"
      when
            System started
      then
            g_lights?.members.forEach(
                  light | light.postUpdate(if(Math::random > 0.5) ON
else OFF)
            )
end

rule "Initialize simulated contact states"
      when
            System started
      then
            g_contacts?.members.forEach(
                  contact | contact.postUpdate(if(Math::random > 0.5)
OPEN else CLOSED)
            )
end

rule "Initialize simulated roller shutter states"
      when
            System started
      then
            g_shutters?.members.forEach(
                  shutter | shutter.postUpdate((Math::random *
10).intValue * 10)
            )
end

rule "Initialize simulated temperature sensors"
      when
            System started or
            Time cron "0 0/1 * * * ?"
      then
            g_temps?.members.forEach(
                  temperature | temperature.postUpdate(25.0 + (25.0 -
(Math::random * 50.0).intValue) / 10.0)
            )
end

rule "Call 'ComingHome' upon bluetooth connection"
      when
            Item bluetooth_device_in_range changed to ON
      then
            callScript("ComingHome")
end
```

Listing 34: Consolidated version of DemoHouse.rules.

```
sitemap DemoHouse label="DemoHouse"
{
      Frame
      {
            Group item=g_corridor label="Corridor" icon="corridor"
            Group item=g_office label="Office" icon="office"
      }
      Frame label="Temperature"  icon="temperature"
      {
            Text item=temp_office_out
            {
                  Frame
                  {
                        Chart item=temp_office_out period=3D refresh=600
service="db4o"
                  }
            }
      }
}
```

Listing 35: Consolidated version of default.sitemap.

Literature

[CronTrigger 2014] Cron Expressions, http://www.quartz-scheduler.org/documentation/quartz-2.1.x/tutorials/tutorial-lesson-06, Version of 16th June 2014

[EnOcean 2014] EnOcean, http://en.wikipedia.org/w/index.php?title=EnOcean&oldid=606446005, Accessed at 30th April 2014

[Harper 2003] Frances K. Aldrich and James Barlow and Keith Cheverst and Karen Clarke, Inside the Smart Home, 2003, Springer

[HomeAutomation 2014] Home automation, http://en.wikipedia.org/w/index.php?title=Home_automation&oldid=612409515, Accessed at 10th June 2014

[InnoQ 2014] openHAB: Home-Automation mit Java, http://www.innoq.com/de/podcast/002-openhab/transcript/, Version of 16th June 2014

[KNX 2014a] KNX (standard), http://en.wikipedia.org/w/index.php?title=KNX_%28standard%29&oldid=611341270, Accessed at 3rd June 2014

[KNX 2014b] KNX Manufacturers list, http://www.knx.org/knx-en/manufacturers/list/index.php, Accessed at 15th June 2014

[MySQL 2014] MySQL, http://en.wikipedia.org/w/index.php?title=MySQL&oldid=613371912, Accessed at 18th June 2014

[OpenHAB 2014] openHAB Wiki, https://github.com/openhab/openhab/wiki, Accessed at 28th May 2014

[OpenSenSe] Open.Sen.se, http://open.sen.se/, Version of 16th June 2014

[OSGi 2011] The OSGi Alliance, OSGi Service Platform - Core Specification, 2011,

[OSGi 2014] OSGi, http://en.wikipedia.org/w/index.php?title=OSGi&oldid=612576910, Accessed at 12th June 2014

[PowerLine 2014] Power-line communication, http://en.wikipedia.org/w/index.php?title=Power-line_communication&oldid=612767736, Accessed at 13th June 2014

[Relational 2014] Relational model, http://en.wikipedia.org/w/index.php?title=Relational_model&oldid=613089044, Accessed at 16th June 2014

[REST 2014] Representational state transfer, http://en.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=613132337, Accessed at 16th June 2014

[TimeSeries 2014] Time series database, http://en.wikipedia.org/w/index.php?title=Time_series_database&oldid=610819116, Accessed at 30th May 2014

[TwistedPair 2014] Twisted pair, http://en.wikipedia.org/w/index.php?title=Twisted_pair&oldid=612727462, Accessed at 13th June 2014

[Wikipedia 2014d] Home automation, http://en.wikipedia.org/w/index.php?title=Home_automation&oldid=612409515, Version of 10th June 2014

[Wireless 2014] Wireless, http://en.wikipedia.org/w/index.php?title=Wireless&oldid=613413055, Accessed at 18th June 2014

[X10 2014] X10 (industry standard), http://en.wikipedia.org/w/index.php?title=X10_%28industry_standard%29&oldid=607926010, Accessed at 10th May 2014

[Xively 2014] Xively, https://xively.com/, Version of 16th June 2014