Institute for Geoinformatics
Course: Web of Things
Lecturers: Dr. Albert Remke, Arne Bröring

Raimund Schnürer
Study program: M.Sc. Geoinformatics
SS 2011

SenseBox Documentation

Architecture and Interfaces

The SenseBox is a water-proof box where different sensors can be attached. The sensors are meant to be arbitrary and flexibly interchangeable. In the scope of this study project, we focused on one ultrasonic sensor and one GPS device. The sensors are connected to an Arduino board which is in turn connected to the main board of the SenseBox. The Arduino[1] board enables to easily access the sensor output. All measurements are then continuously delivered over a defined port. Those are readout with the help of a Java application of the Hardware group. Java requires an operating system to run where we chose Ubuntu. The application of the Hardware group forwards the raw data stream to the application of Transformation group. Additional to the sensor's output values, a timestamp is calculated when the measurement took place. A configuration file defines the ID of each sensor, how many values are sent and in which units they are. These parameters are used by the Transformation group to parse their incoming data stream.

Next, the application of the Transformation group converts the raw data into machine-readable observations. For our study project, this is the traffic flow, the number of cars per minute, and the position of the SenseBox. All those observations are stored in a local PostgreSQL database on the SenseBox. Besides that, the application of the Transformation group generates human-readable announcements which are also saved in the database. The database also contains metadata for the observations and announcements as well as for the transformation models and sensors. Metadata are quite important, for instance to see what a value of a certain observation actually means. Some parameters to adjust the transformation process are stored in properties files on the local CompactFlash[2] card of the SenseBox.

Also on the SenseBox, there is installed a light-weight Tomcat server. There, the RESTful API[3] is deployed as a web application. The RESTful API has access to all observations and announcements, which are stored in the database by the Transformation group. Observations and announcements are converted into XML documents when their URL is invoked. The same applies for the metadata. The Internet connection of the SenseBox is established by a UMTS, or alternatively by a WLAN stick.

[1] http://www.arduino.cc/
[2] http://compactflash.org/about/
[3] http://en.wikipedia.org/wiki/Representational_State_Transfer#RESTful_web_services

Hereby, the installation of a Virtual Private Network (VPN) ensures that the SenseBox's IP-address is static and does not change over time. But not only data can be retrieved, also data can be sent to the SenseBox. For example, users can subscribe to special announcements types. Those subscriptions are stored in the database. Each time a new announcement is generated, the application of the Transformation group goes through the list of subscriptions and informs all subscribers of the particular announcement type via E-Mail, SMS or Twitter. Besides subscriptions, configuration files of transformation models can be uploaded to SenseBox. The RESTful API then creates an additional file containing all names of the changed files. This file is monitored by the application of the Transformation group. Whenever a new file has arrived, the parameters in there need to be verified and changes to the transformation model be applied.

After the SenseBox is switched on, the REST interface sends the ID and the IP of the SenseBox to the SenseBox portal. There, both values are stored in a database. From time to time, the portal checks if the SenseBox is still available. To display all active SenseBoxes on a map, the portal requests also the position of the SenseBox. Based on their last announcement, SenseBoxes are shown in different colours. The portal retrieves all required information via the REST interface of the SenseBox. On the same way, the contact of the SenseBox can be set which shall be contacted in case of an emergency. The Portal is based on the Open Source Geostack[4] bundle consisting of a PostGIS database for storage, GeoExt as a webserver for geographic data, and OpenLayers for visualization purposes. For the registration process, a RESTful API is deployed on the webserver. The RESTful API is programmed in Java whereas the webpages are written in HTML and JavaScript.

Future suggestions

In general, it would be more efficient to have one single web application running on the SenseBox. Thus, the detour via the database could be avoided in some cases and Java objects could be directly exchanged. A nice feature to have are configurable sensors, so that certain parameters (e.g. the measurement frequency) can be remotely modified. Also, the status should be available for each sensor (e.g. off, broken, idle, measuring) and for other components (like battery, UMTS, memory card, CPU). By this, possible error sources could be faster identified. Lastly, the idea to additionally store observations and announcements on the SenseBox portal should be realized. On the one hand, this would relieve direct accesses to the SenseBox, on the other hand it would serve as a data backup, since the SenseBox' storage capabilities are restricted.

[4] http://workshops.opengeo.org/stack-intro/

Database and REST interface

The database

All observations and announcements made by the SenseBox are stored in a local database. We decided to take a PostgreSQL database since first of all it is freely available and secondly we are familiar with it from previous courses. Moreover, it leaves open the possibility to top it up to a PostGIS database. This could be quite helpful in future in case more complex spatial queries shall be performed.

Having a database on board fits also well to the Web of Things concept. The database can be seen as the SenseBox's memory. Compared to traditional sensors in the Sensor Web, which just deliver their data, the SenseBox is more independent. In case of bad Internet connectivity for example, no data will be lost. A database allows also concurrent access for multiple users. This is quite important since observations and announcements are continuously generated and stored in the database. Over the RESTful API, described in the next chapter, those can be also retrieved from outside. The database ensures hereby smooth operations and handles writing and reading conflicts.

The RESTful API

With the help of a so-called RESTful API, data on the SenseBox can be accessed via the Internet. REST is a software design principle and stands for Representational State Transfer. [R1] The goal of REST is to achieve interoperability within a loosely coupled environment. [R2] One of the most famous examples using REST architecture is the World Wide Web. Here and in other REST applications, communication is stateless. This means there is no record of previous interactions and each interaction request has to be handled based entirely on information that comes with it. [R3]

The central concept of RESTful architecture is a resource. [R1] A resource is an abstract concept for any kind of information. In our example, the SenseBox, its observations, announcements, transformation models, and so forth can be considered as resources. Each resource is distinctively identifiable by a URI. In terms of the World Wide Web a URL is usually taken for this purpose. Resources can have different representations, e.g. HTML, XML, JSON or PDF. In our RESTful API, you just can add .json to the URL of a resource to transfer it into JavaScript Object Notation.

Resources in RESTful web services can be manipulated via standardized protocols. In many cases, the Hypertext Transfer Protocol (HTTP) is used. HTTP defines a set of common methods. The most important ones are HTTP GET, POST, PUT and DELETE. GET is used in address bars of all standard web browsers to retrieve a resource (e.g. all observations of the SenseBox). Additional parameters, indicated by a ‘?’ and separated by a ‘&’, help to filter the results (e.g. give me all observations of a certain type in the last hour). The POST method creates a new resource (e.g. sets a new contact for the SenseBox). To update a resource, PUT is used. (e.g. for uploading a changed configuration file of a transformation model). HTTP DELETE removes a resource (e.g. a subscription to a special announcement type).

Currently the REST approach is the most preferred for realizing the goal of web of things. [R2] The main advantage of RESTful web services is their lightweight nature. Compared to Simple Object Access Protocol (SOAP) webservices, they are not as complex, however on the other hand not as powerful. [R1] This minimises the latency and network communication and maximises the independence and scalability of component implementations. [R2] In our study project, only the portal makes use of the RESTful API of the SenseBox. But theoretically, it could be integrated into further applications as well, for example those by Straßen NRW.

Software packages

At the root of the source folder, there are configuration files which can and should be modified that everything works correctly. In the WAR file, those can be found in the folder WEB-INF > classes. In the file “log4j.xml”, you can specify the output folder where the log file is going to be saved:

```
<appender name="RoFi" class="org.apache.log4j.RollingFileAppender">
        <param name="file" value="/home/sense/Sensebox.log" />
```

This is important since the RESTful API provides a GET request for accessing the log file. The other parameters in the “log4j.xml” concern logging settings, like the output format, and do not need to be changed. In the file “REST.properties”, parameters concerning the registration of the SenseBox at the SenseBox portal can be set. For example, the address of the portal can be entered as well as the IP and the ID of the SenseBox. This is necessary that the portal knows under which URL the SenseBox can be reached. In the “DB.properties” file, all settings are stored for establishing a connection to the database. These are for instance the name and the address of the database. Furthermore, there are factors regulating the table maximum size in the database. When this limit is reached, a certain percentage of the data will be deleted. Besides those three configuration files, there is a file called “web.xml” in the WEB-INF folder. This file is essential for initialising the RESTful API. However, it does not have to be modified unless you change the package names or structure.

The properties files are read out in the class “Settings.java” which lies in the package “rest.storage”. Also, in this package, there is the class “Database.java” which establishes the connection to the database. All other classes in this package perform certain queries on the database. Mainly, they select, insert and delete data of a specific resource from the database. The only exception is the class “TransientStore.java” which stores all temporarily data (e.g. the SenseBox contact). Lastly in this package, there is the class “TableOverflowPrevention.java” which checks at regular times the size of the tables “announcement” and “observation”. When a certain size is exceeded, the oldest tuples will be deleted. The “TableOverflowPrevention” threads are started from the class “StartUpRest.java” in the package “sensebox.start”. Also in this class, the registration thread on the portal, described later, is executed.

In the package “rest.resources”, there are specified paths, methods and parameters for each resource in the RESTful API. These are realised by annotations and should be quite self-explanatory. For further information, consult the tutorial “Build a RESTful Web service using Jersey and Apache Tomcat” by IBM [R4]. Not covered there is the class “Configuration.java” which defines the package name of the resources. Also in this class, a mapping between *.xml and *.json resources is established. The package “rest.bean” contains all corresponding objects with their attributes. These objects are used to easily create *.xml and *.json files out of them. The underlying library for this is the JAXB, the Java Architecture for XML Binding.

The package “sensebox.interfaces” was originally created to exchange data between the Hardware, Transformation and REST&Database group. However, due to an interoperability problem for one of the required libraries of the Hardware group, it was not possible to combine the source code of all three groups in a single WAR file. As an alternative, the database was taken as the interface between the groups. The only class in use of this package is “FromRestToPortal.java”. Here, the registration of the SenseBox at the portal takes place. Every time the SenseBox is switched on, it tries to register at the portal as long it gets a positive response. This process is required when either the SenseBox has not got any Internet connection or the portal server is shutdown. The other classes of the package “sensebox.interfaces” are left just in case a unification of the classes from the different groups takes place in future.

Lastly, the package “rest.test” comprises the source code of the tutorials. It also contains two stress tests for the database which rapidly insert data and simulate multiple accesses.

Further materials

Please find below a list of commonly shared resources and tutorials which have been developed within the course by the REST and DB group.

Resources:

- EntityRelationshipModel - Entity-Relationship Model for the SenseBox database
- DatabaseSchema - SQL statements for the database schema on the SenseBox
- SenseBoxMetaData - Metadata about sensors, transformation models, observations and announcements
- HTTPRequestsA - All possible HTTP requests for the SenseBox and the Portal

Tutorials:

- PropertiesTutorial - A tutorial how to save and load settings from a Java .properties file
- DatabaseTutorial - A tutorial how to connect from a Java class to a PostgreSQL database with JDBC
- LoggingTutorial - A tutorial how to log Java classes with log4j
- RESTfulAPITutorial - A tutorial how to set up a RESTful API with Java, Eclipse, Glassfish and Jersey
- RESTfulHTTPClientTutorial - A tutorial how to connect from a Java class to a RESTful webservice with Jersey

Future Ideas

Additional to the JSON and XML representation, resources could be displayed as RSS feeds. Users could then easily subscribe to announcements they are interested in, for example just of a particular road. Also, observations and sensor descriptions could be formatted according to OGC's Observations & Measurement and SensorML specification. This would allow a seamless integration into a Sensor Observation Service, where further analysis functions and display options are available. Experimentally, the Java Persistence API (JPA) could be tested to facilitate database access. At the moment, this is realised with the Java Database Connectivity (JDBC) and involves quite a lot of programming. Lastly, it would be interesting to see whether new transformation algorithms and sensor port listeners could be dynamically uploaded to the runtime of the web application. This would make a remote maintenance possible in case a new sensor is attached to the SenseBox.

References

[R1] Bröring, Arne (2010): Web-Technology. Lecture slides of the course "Spatial Information Infrastructure", University of Münster, pp. 59-63

[R2] Daniel, Desiree (2011): The Future Web: Web Of Things. Paper of the seminar "Web of Things", University of Münster. Online available at: https://wiki.ifgi.de/pub/WOT/Papers/2011-06-01_Pape_Web_of_Things_Desiree_Daniel.pdf

[R3] Borger, Mats and Flanigan, Joseph (2005): What is stateless? Definition from WhatIs.com. Online available at: http://whatis.techtarget.com/definition/0,,sid9_gci213051,00.html

[R4] Huang, Yi Ming et. al (2009): Build a RESTful Web service using Jersey and Apache Tomcat. Technical article by IBM. Online available at: http://www.ibm.com/developerworks/web/library/wa-aj-tomcat/index.html