Cooperative Media Lab

Sens-ation Sensor Infrastructure Developer Documentation

Bauhaus University Weimar
Computer Supported Cooperative Work
Prof. Dr. Tom Gross, Dipl.-Inf. Tareg Egla

Andrea Lahn,
Nicolai Marquardt,
Matthias Pfaff,
Christian Semisch

2004/2005

Abstract:

Sens-ation is a client-server infrastructure for providing sensor information to various clients. With the platform you can implement software applications that can react in dependence on the discovered environment. An example for this scenario is a desktop application that automatically starts your web browser and email client when it has detected that you are in your office.
A major feature of the server infrastructure is providing simultaneous access to sensor information and a common interface for all registered sensor modules. New sensors, hardware modules and locations can be easily registered with the use of XML descriptions. For the notification of new sensor events, the server provides miscellaneous interfaces, e.g. XML transmission with XML-RPC, sockets and HTTP connections as well as a PHP user interface. For the aggregation and interpretation of sensor values, high level services can be integrated as software modules into the infrastructure.
Furthermore we have implemented different example clients for desktop hardware and mobile devices. The clients are implemented in Java as well as AppleScript (Apple scripting language), PHP and J2ME for the mobile client.

Contents

List of Figures

1. Introduction

1.1. What is "Sens-ation"

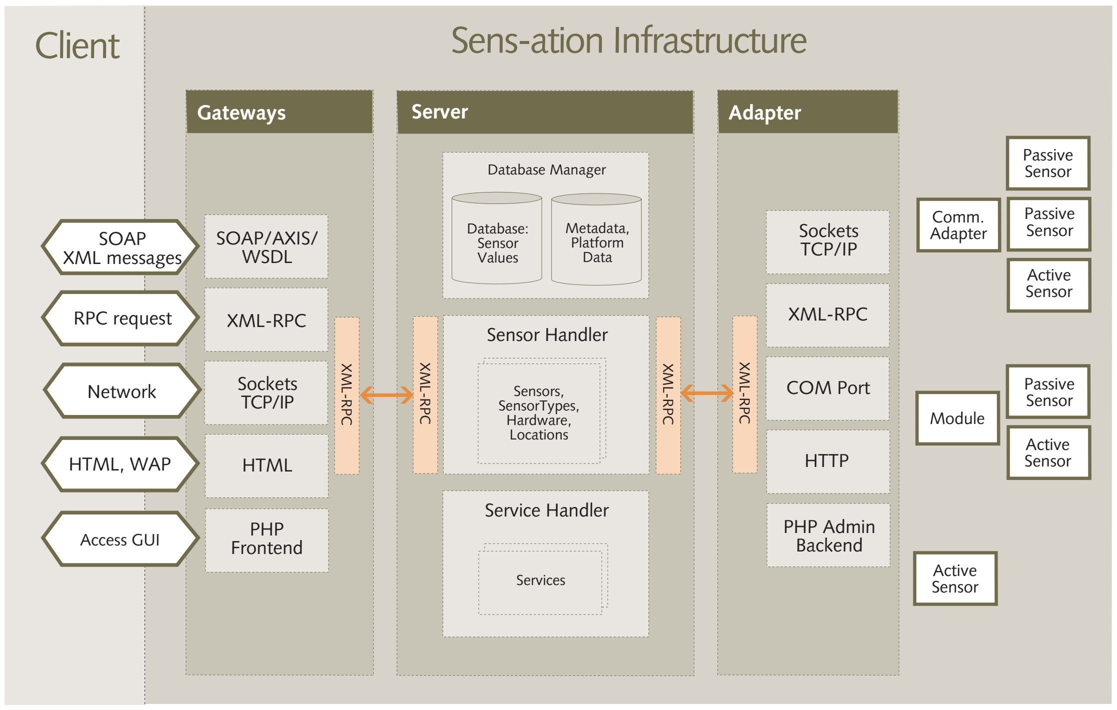


Figure 1.1.: The Sens-ation infrastructure overview

The *Sens-ation Sensor Infrastructure* was one of the CML [Cooperative Media Lab Website] research projects in the winter term 2004/2005. With our client-server infrastructure *Sens-ation* you can implement software tools that react in dependence on the discovered environment. The *Sens-ation* server provides many different interfaces for clients to access the values of sensors connected to the infrastructure. A variety of sensors can connect to the server by implementing a sensor adapter class.

For sensor access it is necessary to protect the sensor hardware from simultaneous access by multiple clients. The solution of this problem is the `SensorHandler` module (chapter 5.1) as virtual interface. The `SensorHandler` is a kind of buffer between the client and used sensor modules, which accumulate and stores sensor information in a database. The gateway modules (chapter 8.1 and 8.2) then provide on the server side a common interface for clients to access all sensor values they need.
A major feature of the server infrastructure is providing simultaneous access to sensor information and a common interface for all registered sensor modules. The sensor adapters (see chapter 5.9) encapsulate and hide all specific implementation details of the hardware (see figure 1.1). New sensors, hardware modules and locations can be easily registered with the use of XML descriptions. And for the notification of new sensor events, the adapter modules provide miscellaneous interfaces, e.g. XML transmission with XML-RPC, sockets or HTTP connections. Beyond the real sensor hardware (current temperature, light intensity or movement) there can be integrated more abstract sensors (mobile sensor, messenger awareness, etc.) as well.
Clients can connect via different gateways: XML-RPC, AXIS Web Services (SOAP), sockets or HTTP connections (chapter 8.1 and 8.2). The clients can communicate in two different ways with the server: they can either connect just for a single value of a sensor or a collection of sensor values, or they register themselves to be notified when events of the specified sensor occur.
For the aggregation and interpretation of sensor values, high level services (chapter 6) can be integrated as software modules into the infrastructure. These services can for example calculate a variety of heuristics from current and saved sensor information. The heuristically calculated values will then also be available through our public service interface.
Furthermore we have implemented different example clients for desktop hardware and mobile devices. These clients demonstrate the easy implementation of client access to the saved sensor values provided by the central server. We have implemented these clients in Java (chapter 9) as well as AppleScript (Apple scripting language, see chapter 9.5), PHP (chapter 8.2) and J2ME for the mobile client (chapter 9.6).

1.2. Features of the Platform

- **Easy integration of sensors:**
  Connect new sensor modules via adapters: section 5.8 and 5.9.

- **Various interfaces for clients:**
  Clients can choose between various access gateways: Thick clients can use e.g. the AXIS/SOAP gateway, scripting languages like AppleScript the XML-RPC gateway, mobile applications the HTML/PHP interface or the client choose the socket TCP/IP gateway: chapter 8.1 and 8.2.

- **XML descriptions:**
  The sensors, locations, sensor types, hardware modules and sensor values are described in XML format. Dynamic XML parser and writer modules enable the easy import and export of the XML descriptions.

- **Push and pull technology:**
  Clients can use the pull technology to contact the server to request sensor values or they subscribe themselves for specific sensors or sensor types (push technology).

- **Flexible exploration and access methods:**
  Various methods for data access: clients can access values in XML format, as string, hashtable, vector etc.

- **Intelligent database:**
  The database storage logic has some filters and algorithms to reduce the amount of data: chapter 7.

- **PHP admin interface:**
  Register new sensors, access sensor values or get CSV file of time series of sensor events: chapter 8.2.

- **Example clients:**
  We have developed example clients for various platforms: Java, AppleScript and J2ME (cell phones): chapter 9.

2. Developer Notes

2.1. Code Conventions

In general we comply with the Java Code Conventions of Sun Microsystems [Java Code Conventions]. These code conventions covers file organization, comments, declarations, statements, naming conventions, programming practices and includes some code examples. We have added many code comments inside the source files to explain the complex sections inside the classes (JavaDoc standard, see [Javadoc Reference]). You can find the *Sens-ation* JavaDoc at [Sens-ation Javadoc]. In the header of each class file you see a short overview about the main tasks of the class.

2.2. Development Tools

For the development we use the Eclipse IDE [Eclipse Foundation] (Version 3.1 M2). This IDE is a open source project of the Eclipse Foundation and beyond the intuitive user interface there exists many plug-in tools to enhance the standard editor.
We recommend the following tools when developing classes for the *Sens-ation* server:

1. **EclipsePHP:**
   Integrated PHP source editor, a HTML preview window and more functionality for the PHP web development. Download the latest release at: `http://sourceforge.net/projects/phpeclipse/`. We recommend to use the release `PHPEclipse1.1.2-2004-12-04`.

2. **EclipseME:**
   Plug-in for development with the Java 2 Micro Edition (J2ME), website

at `http://eclipseme.sourceforge.net/`. MIDlet suite creation, mobile packages, JAD file editor, JAR file packaging and other J2ME development tools.

3. **Wireless Toolkit (version 2.x):**
   A stand-alone state-of-the-art toolbox for developing wireless applications, which contains additional libraries and which can be integrated into Eclipse. Informations and download at `http://java.sun.com/products/j2mewtoolkit/`. Please check `http://eclipseme.sourceforge.net/docs/installation.html` for more installation notes.

4. **For Tomcat:**
   For Tomcat the Sysdeo plugin `http://www.sysdeo.com/eclipse/tomcatPluginV3.zip` is recommended. It can start and stop Tomcat and print the Tomcat log data to the Eclipse console. However, due to version registrations, it will only work with Eclipse 3.0.

5. **For AXIS:**
   Various AXIS plugins for Eclipse exist, some integrated into bigger webservice plugins like JBoss and some smaller seperately working versions, like Andres Aguiar's plugin. However, in this project we used due to compatibility problems the command line version of AXIS.

2.3. Overview of the Packages

- `de.buw.medien.cscw.sensation.server`
  This package contains the main server classes and the handler classes. There are also the sensor, location and sensortype files as well as some server utility classes.

- `de.buw.medien.cscw.sensation.server.hardware`
  The abstract hardware class is used for bidirectional sensor adapter communication. The other files of this package are for the hardware module description.

- `de.buw.medien.cscw.sensation.server.services`
  Services are classes for abstract interpretations of sensor values. A service implementation extends the abstract service class included in this package.

- `de.buw.medien.cscw.sensation.sensors`
  Includes the classes for the ESB communication (open connection, transfer data, execute commands, data collections, string parser).

- `de.buw.medien.cscw.sensation.sensors.adapter`
  The sensor adapter is the connection between hardware sensor modules and the infrastructure (XML-RPC communication).

- `de.buw.medien.cscw.sensation.security`
  In future releases the package should contain the security classes. At the moment this is only the MD5 generator class.

- `de.buw.medien.cscw.sensation.interfaces`
  The interface files for the implementation of the observer pattern [Erich Gamma et al.].

- `de.buw.medien.cscw.sensation.database`
  All classes for the database handling (especially the MySQL database).

- `de.buw.medien.cscw.sensation.client.desktop`
  Client implementations for the desktop environment.

- `de.buw.medien.cscw.sensation.client.midlet`
  MIDlet client for cell phones (for MIDP 1.0).

2.4. Libraries

Overview of the libraries we used (all libraries are located in the `/lib` directory of the source root directory):

- **JDOM:**
  Java Document Object Model API. We use this API for XML parsing process and XML document manipulation as well as the XML string writing. Download and developer documentation at `http://www.jdom.org`

- **MySQL Connector:**
  MySQL Connector/J [MySQL Connector/J] is the database driver. It's necessary for JDBC to get access to the MySQL database

[MySQL Database Website]. This a type 4 JDBC driver, that means the driver is completely in Java implemented and communicate directly with the database server. This is the driver with the best performance.

- **JFreeChart:**
  JFreechart is a library that helps displaying various kinds of charts. It's used in the chart client to display notified events. It can be downloaded at `http://prdownloads.sourceforge.net/jfreechart/jfreechart-0.9.21.zip?download`. It is recommended for rapid development of graphical desktop clients without spending too much time with the implementation of visualizations.

- **XML-RPC:**
  XML-RPC is a library for instantiating clients and servers for accessing methods over a distance by remote calls. In this framework the Apache XML-RPC 1.1 implementation is used `http://archive.apache.org/dist/ws/xmlrpc/v1.1/xmlrpc-1.1.zip`. The source at the location `http://archive.apache.org/dist/ws/xmlrpc/v1.1/xmlrpc-1.1-src.zip` is modified by an authentication patch at the URL `http://www.nighttale.net/OpenSource/download/xmlrpc-authenticated.patch`.

- **kXML-RPC:**
  kXML-RPC is a J2ME implementation of the XML-RPC protocol with an extremely lightweight mechanism for exchanging data and invoking web services in a neutral, standardized XML format (`http://kxmlrpc.objectweb.org/software/downloads/index.html`).

- **AXIS**
  AXIS is a webserver library, which works with the Apache Tomcat webserver application, but also with any other Java webserver application. It uses its own version of XML-RPC to communicate as well as an XML coding for interoperationally transferring data. The webserver is created using functions contained in the AXIS libraries. Learn more at 8.1.3 and at the AXIS webpage at `http://ws.apache.org/axis/news.html`.

2.5. Distribution

To compile the source code and distribute the project you can use the Java-based build tool Apache Ant [Apache Ant] with the `build.xml` file. If you write new Java classes for the *Sens-ation* server, please add these files to the "src.files" property of the `build.xml` file.
You can choose one of the following Ant targets:

- `compile`:
  Compile the source files and write them to the `/build` directory.

- `run`:
  Start the *Sens-ation* server (depends `compile`).

- `dist`:
  Create the `server.jar` file in the `/dist` directory and copy the properties and XML files to this directory too. All required Java 3rd-Party libraries are included in this JAR file. To start the server from commandline just change the path to the `[sensation-source]/dist` path and type "java -jar server.jar".

- `javadoc`:
  Create the JavaDoc [Javadoc Reference] HTML pages in the `/doc` directory.

- `clean`:
  Delete the `/build` and the `/dist` directory.

3. Installation

The following manual pages guides you through the installation process of the *Sens-ation* infrastructure. This package contains various modules: The main server, three gateway modules and sensor adapters. The latter are optional, but at least the main server must be installed on your computer system.

3.1. Requirements

For the main server module:

- **Operating System:** Apple OS X (Version 10.3 or above) or Microsoft Windows (2000 or above)
- **Java Virtual Machine:** Sun Java JRE/JDK, Version 1.4.2

If you want to use one of the additional packages, the following packages are required:

- **Database module:** MySQL database version 4.1.x or above
- **AXIS gateway:** Apache Tomcat server version 5.0.x, AXIS distribution version 1.1
- **PHP/HTML gateway:** Apache server version 2.x, PHP version 4.3.x
- **ESB sensor adapter:** Sun Java Communications API, version 2.0 (only available for Microsoft Windows or SPARC/Solaris). You need this Java extension to access the COM port and establish the communication with the Embedded Sensor Board (ESB) hardware.

The next section will help you to install the missing components of this list for your Apple Mac OS or Microsoft Windows System. If you have already installed all the required components you can skip the following section.
All the required packages are open source software and can be downloaded for free. Often there are various ways to install the required packages, but we explain one installation process that we have tested in detail.

3.2. Installation of Required Components

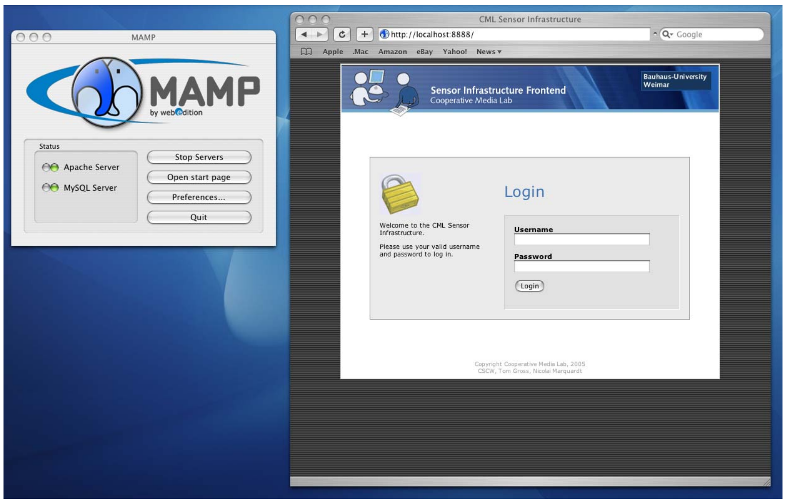
3.2.1. Apple Mac OS X

Java SDK
The Apple Mac OS X operating system has an installed Java Virtual Machine (JVM) by default. You do not need to install any further Java packages.

Apache server, MySQL database and PHP
Instead of installing all the required components seperately, we suggest to download the MAMP[1] installer. This setup will install an Apache server, a MySQL database and the PHP distribution 4.3.x and 5.x on your OS X system. There are no conflicts between the Apache server that perhaps already exists on your system and this new installed server. If you wish to remove this package later, it is only necessary to delete the [Aplications]/MAMP folder.
To install the MAMP package, open the website `http://www.mamp.info/en/home/`, select the download section and load the DMG image file (Sourceforge download site at `http://prdownloads.sourceforge.net/mamp/MAMP_1.0a3.dmg.gz?download`). When the download has finished, just open the DMG file and drag the MAMP folder to the `Applications` directory of your Mac. When starting the MAMP control panel (just doubleclick on the MAMP application file), you can activate and deactivate the apache server and the MySQL database (see figure 3.1).

If you want to use the graph visualizations of the PHP interface, you also have to install the `Image_Graph` package of the PEAR project (`http://pear.php.net/package/Image_Graph`). To install the graph library download the source package (0.3.0dev4, `http://pear.php.net/get/Image_Graph-0.3.0dev4.tgz`) and

[1]MAMP = Macintosh Apache MySQL PHP



Welcome to the CML Sensor Infrastructure.

Please use your valid username and password to log in.

Copyright Cooperative Media Lab, 2005
CSCW, Tom Gross, Nicolai Marquardt

Figure 3.1.: The MAMP server startup panel

extract the files to the `[PEAR]/Image/Graph/` directory. See the PEAR website (`pear.php.net`) for more information about installing the PEAR packages.

Apache AXIS and Tomcat

This installation is only required if you want to use the AXIS Gateway of the package. Additionally to the server installation, you'll need the following software. Download it by clicking on the links.

- Tomcat 5.0.28
- AXIS 1.1
- Xerces 2.5.0
- XStream-1.0.2
- JDOM 1.0
- XML-RPC 1.2b1

Download Tomcat to your desktop, otherwise the quick starting of Tomcat with the shellscripts will not function. Download the AXIS 1.1 Final package and unzip it. Copy the `webapps` folder from your AXIS folder to the `webapps` folder in your Tomcat installation folder. Download the

Xerces XML-parser. Copy the `xercesImpl.jar`, `xercesSamples.jar`, `xml-apis.jar`, `xmlParserAPIs.jar` from the unzipped Xerces folder to the `axis-1_1/lib` folder. Download `XStream-1.0.2` and `JDOM 1.0` and copy the files `xmlrpc1.2b1.jar`,`xstream1.0.2.jar` and `jdom.jar` to the `axis-1_1/libs` folder. Also add a copy of all files in the `axis-1_1/libs` folder to the `[your Tomcat folder]/webapps/axis/WEB-INF/lib/` folder . Make sure that all files that are now in the `axis-1_1/libs` folder are set in the `CLASSPATH` environment variable. You can set the classpath by typing `export CLASSPATH=[filetobeadded]`, where `[filetobeadded]` is replaced by the file that you want to add. Also `export` the `JAVA_HOME` variable, which should contain the path of your Java JDK installation. Copy all the libs that are now in the folder to the `[your Tomcat folder]/webapps/axis/WEB-INF/lib/` folder.

Alternatively, take the `axislib` folder from the package you just downloaded and copy it to your desktop. It contains all the needed libraries that are listed above. Place a copy of all the libs in `[your Tomcat folder]/webapps/axis/WEB-INF/lib/`. Make sure that you have rights for execution of all files that you installed by typing `chmod -R 777 *` in the `Tomcat` folder, else an error will occur telling you that permission of access is denied. Copy both the `axisserver.sh` and the `axisservershutdown.sh` to the desktop. Edit the entry for `DESKTOP_PATH` to the path your desktop is running on, and the `JAVA_HOME` entry to your JDK home directory, using your favourite text-editor. Execute the `axisserver.sh` file by typing in the terminal in your desktop path `./axisserver.sh`. This should add any required libraries to the `CLASSPATH`, and start Tomcat. However, also, you will have to add the libs for the whole session. To do this, just copy and paste the command line instructions from the `axisserver.sh` file to your terminal (change the paths as described above before copying). Otherwise, the deployment of your service will not work, because the environment variables are deleted after the execution of the shell script. To shutdown the webserver, just execute the `axisservershutdown.sh` shellscript.

Java Communication API

The Java Communications API is only available for Microsoft Windows and the Sparc/Solaris platform. This means, that the *Sens-ation* adapter for the Embedded Sensor Board (ESB) can not be executed on Mac OS X systems.

3.2.2. Microsoft Windows

Java SDK
At the Sun website you can find the Java packages as runtime edition (JRE) or the development kits (JDK): open the website http://java.sun.com/j2se/1.4.2/download.html and select the JRE or JDK download. Accept the licence agreement and select your Operating System (Windows). Execute the setup programm and follow the guided installation process. After installation you must reboot your system.

Apache server, MySQL database and PHP
We suggest to download the complete installation package of the XAMPP project. This package contains all required components for the *Sens-ation* PHP gateway.
To install the XAMPP package, open the website http://www.apachefriends.org/en/xampp-windows.html, select the installer setup (Sourceforge download site) and follow the installation steps of the setup wizzard.
If you want to use the graph visualizations of the PHP interface, you alos have to install the `Image_Graph` package of the PEAR project (`http://pear.php.net/package/Image_Graph`). To install the graph library download the source package (0.3.0dev4, `http://pear.php.net/get/Image_Graph-0.3.0dev4.tgz`) and extract the files to the `[PEAR]/Image/Graph/` directory. See the PEAR website (`pear.php.net`) for more information about installing the PEAR packages.

Apache AXIS and Tomcat
This installation is only required if you want to use the AXIS Gateway of the package. Additionally to the server installation, you'll need the following software. Download it by clicking on the links.

- Tomcat 5.0.28
- AXIS 1.1
- Xerces 2.5.0
- XStream-1.0.2
- JDOM 1.0
- XML-RPC 1.2b1

Download Tomcat 5.0.28. Follow the installation instructions in the installer. Be sure to delete alle whitespace from the installation path. On the port dialog, type `8080`. In the final installation window, uncheck the option of starting the Tomcat server. Download the AXIS 1.1 Final package and unzip it. Download the Xerces XML-parser. Copy the `xercesImpl.jar`, `xercesSamples.jar`, `xml-apis.jar`, `xmlParserAPIs.jar` from the unzipped xerces folder to the `axis-1_1/lib` folder. Copy the `webapps` folder from your AXIS folder to the `webapps` folder in your Tomcat installation folder. Download `XML-RPC 1.2b1`, `XStream-1.0.2` and `JDOM 1.0` and copy the files `xmlrpc1.2b1.jar`,`xstream1.0.2.jar` and `jdom.jar` to the `axis-1_1/libs` folder. Make sure that all files that are now in the `axis-1_1/libs` folder are set in the `CLASSPATH` environment variable. To put files to an environment variable, click on `Start->System->My Computer->Advanced->Environment variables->New`, and add the name of the variable you want to initialize and the path of the files you want to include. Copy all libs that are now in the folder to `[your Tomcat folder]/webapps/axis/WEB-INF/lib/`.

Alternatively, take the `axislib` folder from the package you just downloaded and copy it to your drive C: AND to `[your Tomcat folder]/webapps/axis/WEB-INF/lib/`.
Execute the `axisinit.bat` file that is also in the folder in a console. This should add any required libraries to the temporary `CLASSPATH` in your console, enabling you to deploy AXIS webservices.

Java Communication API
You can download the Java Communication API install package at the website http://java.sun.com/products/javacomm/downloads/index.html and install the software at your system or you can download the "ESB adapter" with a bundled JAVA runtime edition from the CML website [Sens-ation Download]. With this complete package you do not need any further components.

3.3. Installation of the Server

3.3.1. Apple Mac OS X, UNIX

1. **Verify** that all the required packages (see 3.1) are installed on your system and if necessary install the missing components (see 3.2.1).

2. **Download** the *Sens-ation OS-X binary package* from the CML website at [Sens-ation Download].

3. **Extract** the compressed ZIP file to the location on your harddrive where you want to save the server executables.

4. **Decide** if you want to use the MySQL database or the pre-configured XML sensor and location descriptions. By default, the database is disabled and the server will load the descriptions from the XML files instead. If you want to change this or one of the other settings, see section 3.3.3

5. **Start** the server: double-click the server.jar file in the Finder window (OS X) or open a terminal window, change the path to the directory you have saved the server binary files and there type `java -jar server.jar` to start the server.

6. **Server** console (see figure 3.2): you can see the startup sequence: initialization of the gateway modules, the sensor handler, registration of sensors, locations and services. When the initialization has finished, you see the server console prompt. Congratulations! The *Sens-ation* server is now ready.

3.3.2. Microsoft Windows

1. **Verify** that all the required packages (see 3.1) are installed on your system and if necessary install the missing components (see 3.2.2).

2. **Download** the *Sens-ation Windows binary package* from the CML website at [Sens-ation Download].

3. **Extract** the compressed ZIP file to the location on your harddrive where you want to save the server executables.

4. **Decide** if you want to use the MySQL database or the pre-configured XML sensor and location descriptions. By default, the database is disabled and the server will load the descriptions from the XML files instead. If you want to change this or one of the other settings, see section 3.3.3

5. **Start** the server: double-click the server.exe file in the Windows Explorer or open system console (Start → Execute... → type "cmd" and press enter),

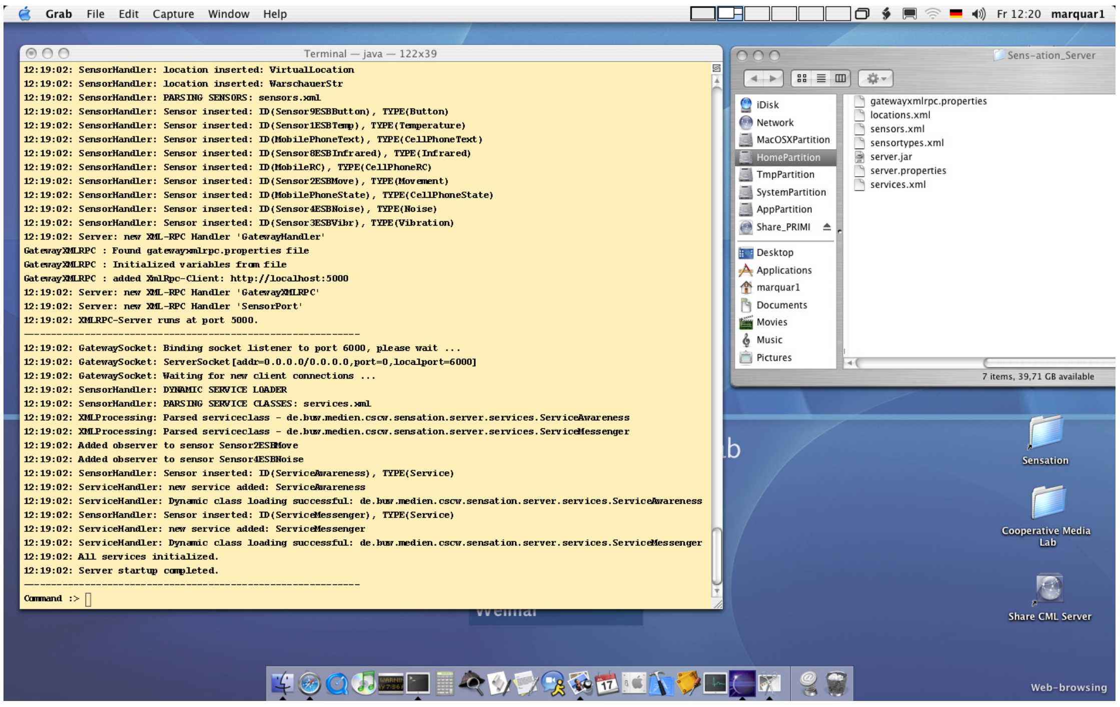

```
12:19:02: SensorHandler: location inserted: VirtualLocation
12:19:02: SensorHandler: location inserted: WarschauerStr
12:19:02: SensorHandler: PARSING SENSORS: sensors.xml
12:19:02: SensorHandler: Sensor inserted: ID(Sensor9ESBButton), TYPE(Button)
12:19:02: SensorHandler: Sensor inserted: ID(Sensor1ESBTemp), TYPE(Temperature)
12:19:02: SensorHandler: Sensor inserted: ID(MobilePhoneText), TYPE(CellPhoneText)
12:19:02: SensorHandler: Sensor inserted: ID(Sensor8ESBInfrared), TYPE(Infrared)
12:19:02: SensorHandler: Sensor inserted: ID(MobileRC), TYPE(CellPhoneRC)
12:19:02: SensorHandler: Sensor inserted: ID(Sensor2ESBMove), TYPE(Movement)
12:19:02: SensorHandler: Sensor inserted: ID(MobilePhoneState), TYPE(CellPhoneState)
12:19:02: SensorHandler: Sensor inserted: ID(Sensor4ESBNoise), TYPE(Noise)
12:19:02: SensorHandler: Sensor inserted: ID(Sensor3ESBVibr), TYPE(Vibration)
12:19:02: Server: new XML-RPC Handler 'GatewayHandler'
GatewayXMLRPC : Found gatewayxmlrpc.properties file
GatewayXMLRPC : Initialized variables from file
GatewayXMLRPC : added XmlRpc-Client: http://localhost:5000
12:19:02: Server: new XML-RPC Handler 'GatewayXMLRPC'
12:19:02: Server: new XML-RPC Handler 'SensorPort'
12:19:02: XMLRPC-Server runs at port 5000.
------------------------------------------------------------
12:19:02: GatewaySocket: Binding socket listener to port 6000, please wait ...
12:19:02: GatewaySocket: ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=6000]
12:19:02: GatewaySocket: Waiting for new client connections ...
12:19:02: SensorHandler: DYNAMIC SERVICE LOADER
12:19:02: SensorHandler: PARSING SERVICE CLASSES: services.xml
12:19:02: XMLProcessing: Parsed serviceclass - de.buw.medien.cscw.sensation.server.services.ServiceAwareness
12:19:02: XMLProcessing: Parsed serviceclass - de.buw.medien.cscw.sensation.server.services.ServiceMessenger
12:19:02: Added observer to sensor Sensor2ESBMove
12:19:02: Added observer to sensor Sensor4ESBNoise
12:19:02: SensorHandler: Sensor inserted: ID(ServiceAwareness), TYPE(Service)
12:19:02: ServiceHandler: new service added: ServiceAwareness
12:19:02: ServiceHandler: Dynamic class loading successful: de.buw.medien.cscw.sensation.server.services.ServiceAwareness
12:19:02: SensorHandler: Sensor inserted: ID(ServiceMessenger), TYPE(Service)
12:19:02: ServiceHandler: new service added: ServiceMessenger
12:19:02: ServiceHandler: Dynamic class loading successful: de.buw.medien.cscw.sensation.server.services.ServiceMessenger
12:19:02: All services initialized.
12:19:02: Server startup completed.
------------------------------------------------------------
Command :> 
```

Figure 3.2.: Sens-ation server started, console window

change the path to the directory you have saved the server binary files and there type "server.exe" to start the server.

6. **Server** console (see figure 3.2): you can see the startup sequence: initialization of the gateway modules, the sensor handler, registration of sensors, locations and services. When the initialization has finished, you see the server console prompt. Congratulations! The "Sens-ation" server is now ready.

3.3.3. Properties File

The `server.properties` file is the configuration file for the server and database module. It is located in the root directory of the extracted server package. Each configuration variable in the file is annotated to explain the effects when changing the entry. If you have changed the `server.properties` file and you want to restore the original file: delete the `server.properties` file and rename the file `server.default.properties` to `server.properties`.

By default the database is disabled and the server will start without using a database (the internal cache of the server is used instead). To enable the database open the `server.properties` file in a texteditor (e.g. TextEdit or Notepad) and follow the changes in chapter 3.4.1.
The properties file has two main parts: PART A (at the beginning of the file) is for the main changes to enable the databse usage. PART B is for further changes of the server configuration. Please read the comments in the config file for more information about the entries (see chapter A.2.1).

3.4. Installation of Optional Packages

3.4.1. Database

The MySQL database and the server needn't to be installed on the same computer. If the computers are connected to each other, it will also work distributed.
Before you can use the database with our framework, you have to do some things:

1. Create a new MySQL accound for the framework. Take care that the user have the rights for creating a database and tables.

2. Change some entries in the `server.properties` file:

    - Set the property "server.usedatabase" to true.

    - Change the user name and password in this file: "databse.user" and "database.password"

    - Change the host address of the database in this file: "database.hostIP"

    - Perhaps change the names of the database or tables which will be created by our framework.

    - If the server at starting up should read the registered location and sensors from the database change the status of "server.initfromdatabase" to true.

3. Take care, if the database is running, if you want to use it.

3.4.2. AXIS Gateway

Mac OS

1. **Start Tomcat** with your `axisserver.sh` shellscript by typing `./axisserver.sh` in the terminal in your desktop folder. The entries in the script file have to be edited, and the `CLASSPATH` must be set to function (see 3.2.1). Also, you may start Tomcat by moving to the `bin` folder of your Tomcat installation and typing `./startup`.

2. **Verify** the installation of Tomcat and AXIS by opening `http://127.0.0.1:8080/axis/` in your browser . There should be a site telling you that AXIS is running.

3. **Stop Tomcat** again, by either typing `./axisservershutdown.sh` in the desktop console or in the `bin` folder of the Tomcat installation typing `./shutdown.sh`

4. **Copy** the `axisserver/ws` source package from the package you just downloaded and copy it to [yourtomcatfolder]/webapps/axis/WEB_INF/classes/(axisserver/ws).

5. **Copy** the `axisserver.properties` file to the `[yourtomcatfolder]/bin directory`

6. **Change** the IP of the entry `gatewayhandler.ip` and `gateway.xmlrpc.port` in the `axisserver.properties` file to the IP and the port of the computer the `GatewayHandler` is running on. Change the entries for `gatewayaxis.ip` and `gatewayaxis.port` to the computer on that the AXIS gateway should be running.

7. **Start Tomcat** with your `axisserver.sh` shellscript by typing `./axisserver.sh` in the terminal in your desktop folder. The entries in the script file have to be edited, and the `CLASSPATH` must be set to function (see 3.2.1). Also, you may start Tomcat by moving to the `bin` folder of your Tomcat installation and typing `./startup`.

8. **Deploy** the service typing `java org.apache.axis.client.AdminClient axisserver/ws/deploy.wsdd` in a terminal in the `[yourtomcatfolder]/webapps/axis/WEB-INF/classes`. The AXIS libs must be set in the `CLASSPATH` to work.

9. **Verify** the webservice installation by opening http://127.0.0.1:8080/axis/servlet/AxisServletn your browser to see if an entry exists labeled "AxisServer" (See 3.3).

The Sensation AXIS Webserver is now installed and can be reached by the address http://[Axisserverip]:[TomcatPort]/axis/services/AxisServer.

Microsoft Windows

1. **Start Tomcat** by either starting the Tomcat Monitor from your Startmenu (`Start->Apache Tomcat 5.0->Monitor Tomcat`) and right-clicking on the Tomcat Monitor in the shortcuts menu, and then clicking `Start Service` or double click `startup.bat` in the `bin` folder of your Tomcat installation.

2. **Verify** the installation of Tomcat and AXIS by opening `http://127.0.0.1:8080/axis/` in your browser . There should be a site telling you that AXIS is running.

3. **Stop Tomcat** again by either clicking right on the Tomcat Monitor and then clicking `Stop Service` or double clicking `shutdown.bat` in the `bin` directory of your Tomcat installation.

4. **Copy** the `axisserver/ws` source package from the package you just downloaded and copy it to `[yourtomcatfolder]/webapps/axis/WEB_INF/classes/(axisserver/ws)`.

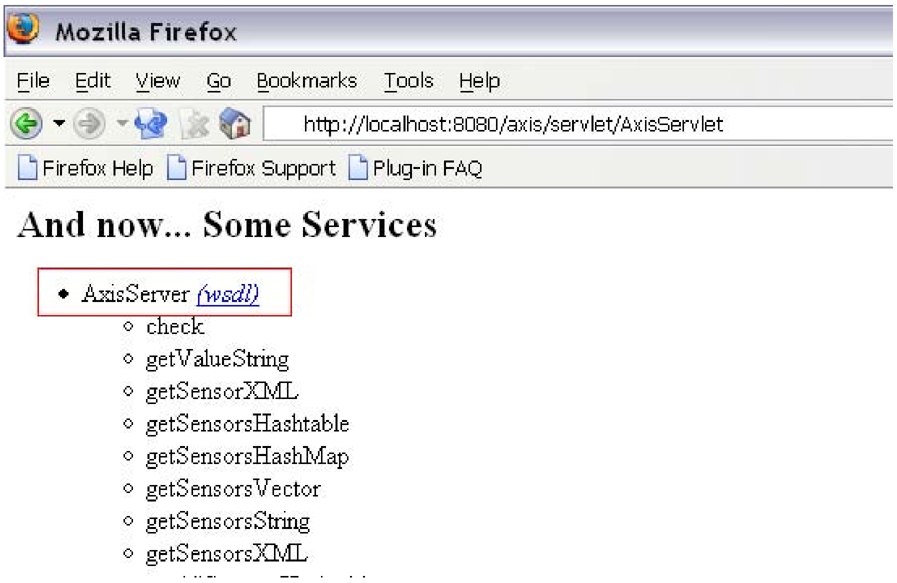
5. **Copy** the `axisserver.properties` file to the `[yourtomcatfolder]/bin directory`

6. **Change** the IP of the entry `gatewayhandler.ip` and `gateway.xmlrpc.port` in the `axisserver.properties` file to the IP and the port of the computer the `GatewayHandler` is running on. Change the entries for `gatewayaxis.ip` and `gatewayaxis.port` to the computer on that the AXIS gateway should be running.

7. **Start Tomcat** by either by right-clicking on the Tomcat Monitor in the shortcuts menu, and then clicking `Start Service` or double click `startup.bat` in the `bin` folder of your Tomcat installation.

8. **Deploy** the service typing `java org.apache.axis.client.AdminClient axisserver/ws/deploy.wsdd` in a terminal in the `[yourtomcatfolder]/webapps/axis/WEB-INF/classes`. The AXIS libs must be set in the `CLASSPATH` to work.

9. **Verify** the webservice installation by opening `http://127.0.0.1:8080/axis/servlet/AxisServlet` in your browser to see if an entry exists labeled "AxisServer" (See 3.3).


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/axis/servlet/AxisServlet
Firefox Help Firefox Support Plug-in FAQ

And now... Some Services

- AxisServer (wsdl)
  - check
  - getValueString
  - getSensorXML
  - getSensorsHashtable
  - getSensorsHashMap
  - getSensorsVector
  - getSensorsString
  - getSensorsXML

Figure 3.3.: AXIS gateway installed properly

The Sensation AXIS Webserver is now installed and can be reached by the address http://[Axisserverip]:[TomcatPort]/axis/services/AxisServer.

3.4.3. PHP Gateway

1. The files for the PHP gateway are in the *Sens-ation* server binary file [Sens-ation Download].

2. Copy the contents of the `[sensation-server]/php-gateway/` directory to the `htdocs` directory of your apache webserver:
    - OS X: You find the `htdocs` directory in `Applications/MAMP/htdocs/`. See figure 3.4.
    - Windows: You find the `htdocs` directory in `[XAMPP-install-directory]/htdocs/`
3. Open the file `config.inc.php` and change the server destination (if you have installed the server on the local computer, just insert `localhost`). (see figure 3.5)
4. Open the file `admin.inc.php` and change the superuser name and password.
5. Start your webbrowser, type the apache directory (OS X/MAMP: `localhost:8080`, Windows: `localhost`) in the browser address bar and enter your superuser login.
6. Now you can select one of the menu entries to access the sensor values.

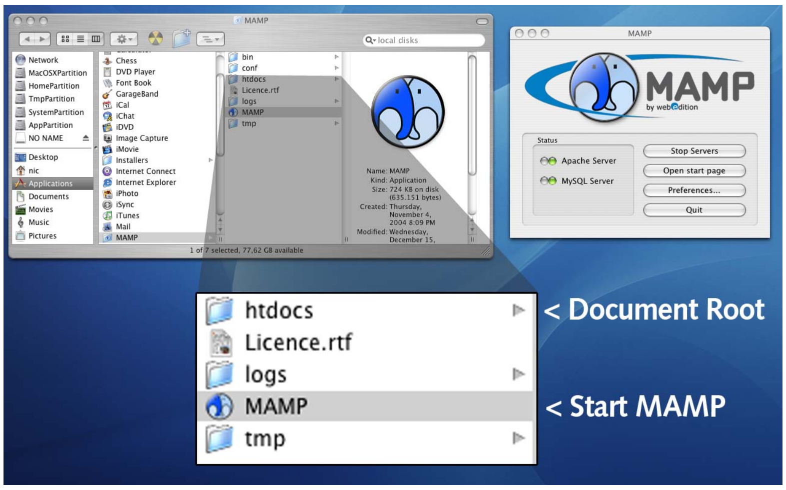
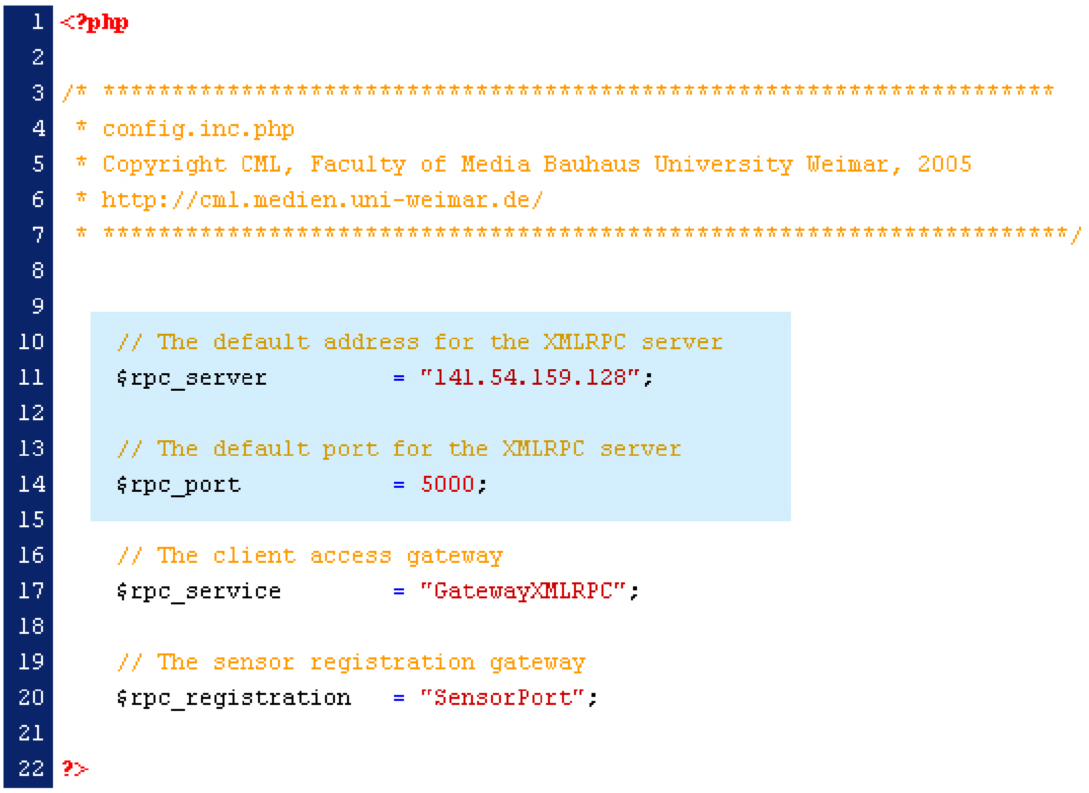


Figure 3.4.: HTDOCS directory of the MAMP server



```
<?php

/* ************************************************************************
 * config.inc.php
 * Copyright CML, Faculty of Media Bauhaus University Weimar, 2005
 * http://cml.medien.uni-weimar.de/
 * ************************************************************************/


    // The default address for the XMLRPC server
    $rpc_server         = "141.54.159.128";

    // The default port for the XMLRPC server
    $rpc_port           = 5000;

    // The client access gateway
    $rpc_service        = "GatewayXMLRPC";

    // The sensor registration gateway
    $rpc_registration   = "SensorPort";

?>
```

Figure 3.5.: PHP configuration file `config.inc.php`

3.4.4. ESB Sensor Adapter

The ESB sensor adapter is the connection module of the ESB hardware to the *Sens-ation* server. It opens a COM connection to the ESB module (COM1 or COM2) and a XML-RPC connection to the server to publish the new sensor events.
Note: This adapter has to be installed on a system with the Microsoft Windows operating system, because the Java Comm API is only available for this platform.

1. Download the *ESB sensor adapter package* from the CML download website [Sens-ation Download].

2. Extract the ZIP file to the desired location on your HDD.

3. Connect the ESB module to the COM1 or COM2 port.

4. Open a terminal software (like *TerraTerm* or *HyperTerminal*), select the

COM port you have connected the board and choose the following settings: Baud rate: 115.200, Data bits: 8 bit, Parity bit: none, Stop bit: 1 bit, Flow control: none

5. Enable the *local echo* to see your console input.

6. Then start the serial port connection

7. Type `saf 32` and press Ctrl+Enter. The sensor board should now send the event string each second.

8. Close the terminal software.

9. Connect the optional second ESB module to the COM port, start the terminal software again and configure the software as listed above.

10. Type `saf 16` and press Ctrl+Enter. This enables the wireless transmission mode, so you can use this ESB module without a connection to the PC.

11. Disconnect this ESB module and connect the first ESB module again.

12. Open a new console window (Win 2000: "Start", "Execute", type "cmd" and press enter) and change the path to the directory with the *ESB sensor adapter* binarys.

13. Type: "`esb [COM] [IP] [PORT]`", where [COM] is the COM port you have connected the ESB module, the [IP] and [PORT] are the address to the running Sens-ation server, so you can for example write: "`esb COM1 localhost 5000`".

14. *Note: You can also change the* ***`start-esb.bat`*** *file to change the settings permanent.*

15. If everything was correct, you can see the ESB messages and XML notification messages scrolling down the screen.

4. Server Architecture

4.1. Modules

The `Server` class is the main part of the *Sens-ation* infrastructure. It initializes the SensorHandler, ServiceHandler and `GatewayHandler`, starts the socket thread, the Apache webserver component and registers the XML-RPC gateways (see figure 4.1).

SensorHandler	Is responsible for the management of all sensors, locations and sensor types. Has various methods for exploration as well as getter and setters. (more details: section 5.1).
GatewayHandler	The connection class for all developed gateways: Access via the XML-RPC connection (port 5000), includes method for location and sensor discovery, registration of clients for sensor events and get methods for SensorValues (section 4.4).
ServiceHandler	Responsible for the service classes. Dynamic class loading of the classes specified in the services.xml file. More details in chapter 6.

4.2. Utility Classes

- `XMLProcessing`
  This module has several methods for parsing (method names with: `parse[type]XML`, e.g. `parseSensorsXML`) and writing (method names with: `get[type]XML`, e.g. `getSensorsXML`) XML data. While the single parsing process of data (locations, sensors, etc.) is done by the objects themself (with the `parseXML` method), the `XMLProcessing` method can parse collections of

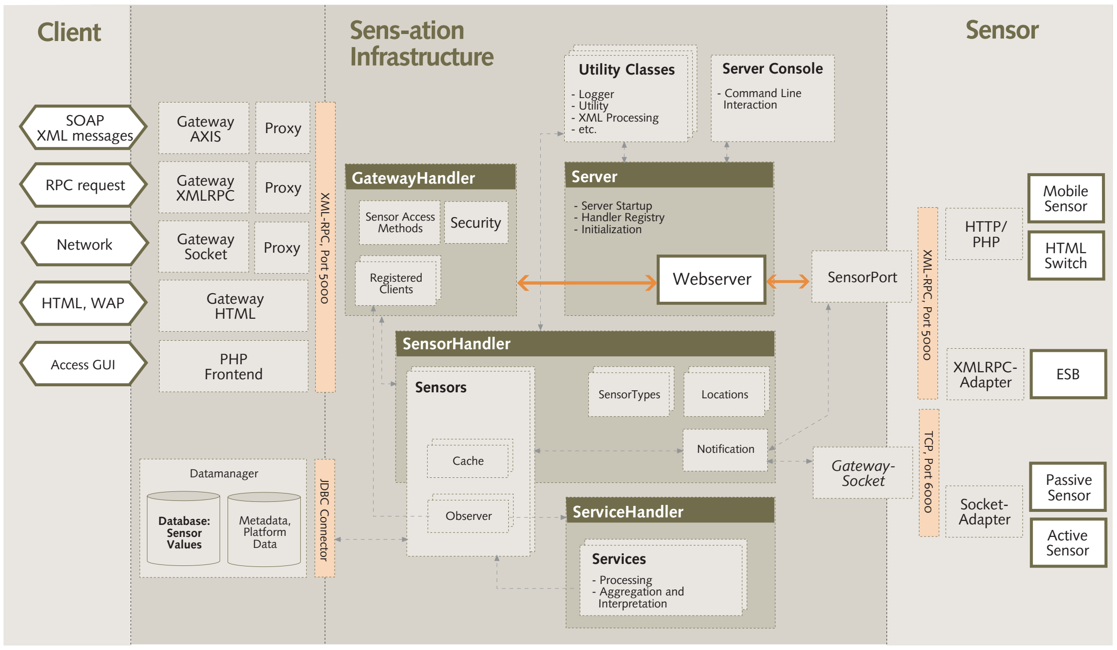


Figure 4.1.: The Sens-ation infrastructure architecture details

data and return ArrayLists with the objects. The `SensorHandler` parses the sensor configurations of the `sensors.xml` file and uses the `parseSensorsXML` method to get a list of all parsed sensors (to insert them into the hashtable).

- `Logger`
  The `Logger` is responsible for writing the console output (e.g. info messages, error messages, state description, etc.) of the server and the connected modules to the system console window. All methods of the `Logger` are static and can be used without creating a instance of the Logger object.
  It is conceivable to extend this module to write the messages to a log file, or to display the messages in different GUI (swing) windows.

- `Utility`
  The methods `stringToDate` can parse a string to a Java date object and the `dateToString` method writes the parameter date time to a string. The date and time format inside the "Sens-ation" platform and for input/output follows the rules of the ISO 8601 guideline [ISO 8601 Date and Time]. The format is "yyyy-MM-dd hh:mm:ss" (where `yyyy` is the year with four digits, `MM` the month with two digits, `dd` the day with two digits, `hh` the hours passed since midnight (0-23), `mm` the minutes and `ss` the seconds).

- `InitializeCML`
  The methods of this class can initialize the standard CML locations, sensors and services we used during the development process. By default the server will load the descriptions from the XML files in the server root directory, but with changes in the `server.properties` file you can changes this setting: set all three `server.initializecml.x`, properties to true and delete the lines with `server.sensorhandler.sensorfile/locationfile/sensortypefile`.

4.3. Console Interaction

The console thread (class: `ServerConsole`) started by the server class is for command line interaction with the server module. The module uses the reflection API to use the methods of the commands class to execute the commands by invoking the corresponding method. The code fragment for dynamic method execution can be found in listing 4.1. We use the commands class (line 3) to search for the corresponding method (line 6) of the command string (line 1) with a vector as parameter (line 4-5). Then the argument vector (line 9) and the commands class are used to invoke the method (line 11).

Listing 4.1: Using the reflection API to execute commands, Class: ServerConsole (without try/catch blocks)

```
1  String command = "list sensors";
2  Commands com = new Commands();
3  Class commandClass = com.getClass();
4  Class partypes[] = new Class[1];
5  partypes[0] = Vector.class;
6  Method commandMethod = commandClass.getMethod(command, partypes);
7
8  Object arglist[] = new Object[1];
9  arglist[0] = params;
10 Object returnValue = null;
11 returnValue = commandMethod.invoke(com, arglist);
```

In the server condole window you can enter various commands to display sensor, location and hardware information as well as some XML descriptions and other informations. You can see the command prompt "Command" that means that you can type your command.

The following commands can be executed (equivalent to the methods of the class "Commands"):

- *list sensortypes*
  Print all registered sensor types (parsed from the sensortypes.xml file). These are the main categories of sensors, e.g. temperatur, movement, light intensity, etc.

- *list sensors—loactions—services*
  Print all registered sensors/locations/services to the console window.

- *delete [sensorID—locationID]*
  Delete the sensor (with sensorID) or the location (with locationID) from the hashtable of the sensorHandler.

- *property [name]*
  Print value of a property from cmlserver.properties file to console.

- *value [sensorID]*
  Get the last value of the sensor. Uses the sensorValue.toString() method.

- *exit*
  Closes all registered webserver handler and shutdown the server.

- *xml location [locationID]*
  Print the location with the given locationID as XML description.

- *xml sensor [sensorID]*
  Print the sensor with the given sensorID as XML description.

- *xml sensors—locations—sensortypes*
  Print the complete XML description of the sensors, locations or sensor types.

- *server rc*
  Print all registered notification clients (in the notification table of the `GatewayHandler` class).

- *db write sensors—locations*
  Write all registered sensors/locations to the connected database.

- *db on—off [sensorID]*
  Activate or deactivate the boolean tag for writing the sensor values to the database. By default all these tags of the sensor objects are set to false, so for history value access only the internal cache of the sensor objects (100 values) will be used.

If you want to implement additional commands, just add the method to the `Commands` class, so that they can be found with the reflection API call of the `SensorConsole` class.
Instead of using only the local console for user interaction, it is also conceivable to implement a remote server control, e.g. with the use of a TCP socket connection. This would be very useful for remote administrator access to the server module. Due to the implementation of the `Commands` class as command encapsualtion this extension as well as other server interaction methods can easily be implemented.

4.4. GatewayHandler

The `GatewayHandler` class represents the main node between the gateways, which are responsible for building a public interface to external access, and the SensorHandler, whose task is to manage sensors. All gateways connect to this class to process requests to the sensor handler. It is an XML-RPC server to all gateways for incoming requests, and a client for outgoing calls. Also, it contains the main mechanism for user registry to sensors or services for XML-RPC and for AXIS webservices. This means, that users may register for a distinct sensor and are afterwards automatically notified if an event occured on that sensor. For example, if a user registers for a movement sensor, and, because of movement in the room, the sensor fires an event, then the user receives a message telling him that something is moving. To manage that feature, the GatewayHandler implements the observer pattern as subject to the SensorHandler. The main user registry consists of a simple hashmap, containing another hashtable for every sensor that has at least one interested user. Everytime a user registers, it is checked, if already a hashtable with the sensor's name as key exists in the hashmap. If that is true, the user is casted upon a RegEntry class and added to the sensor's hashtable. Otherwise, a new hashtable is created and added to the registry with the sensor's name as key, and the register method on the sensor handler is called.

```java
public synchronized String register(String ip, String sensorID, String port) {
        ip=ip.trim();
        sensorID=sensorID.trim();
        port=port.trim();
        if ((ip == null) || (sensorID == null) || (port == null)
                || (ip.equals("")) || (sensorID.equals("")) || (port.equals("")
    )) {
            // if one of the params is empty, report error
            Logger.message("Error registrating Gateway Client IP: " + ip
                    + " SensorID " + sensorID + " on port " + port);
            return "error";
        } else {
            RegEntry re = new RegEntry(ip, port, sensorID);
            // a new RegEntry is created
            // when already a hashtable exists having the sensorID as key,
            // the client is just added to it
            if (clientRegister.containsKey(sensorID)) {
                Hashtable ht = (Hashtable) clientRegister.get(sensorID);
                ht.put(ip, re);
                clientRegister.put(sensorID, ht);
                //when no hashtable with the key of the sensorID exists
                //a new one is created, and the RegEntry put in
            } else {
                Hashtable ht = new Hashtable();
                ht.put(ip, re);
                clientRegister.put(sensorID, ht);
                registerService(sensorID);
            }
            Logger.messageComm("Gateway Client " + ip + " is registrating for "
                    + sensorID + " on port " + port);
            return "done";
        }
    }
```

This is only happening at this point of time, because the sensor handler, unlike the gateway handler, only holds one entry in its registry for every sensor, that at least one user wants to be notified of. You can print both the registered XML-RPC- and AXIS clients by typing on the server console `server rc`. Unregistering is much the same procedure as registering, also deleting empty sensor hashtables. Similarily as being a subject to the SensorHandler class, the gateway handler is also observer to the registered clients, using XML-RPC communication. The notify method informs users about events from the sensors using XML-RPC. Everytime a notify method is called, the registry is checked if there is an entry of the sensor‘s name in ther user registry. If that is the case, the sensor’s user hashtable is mapped to its RegEntries which consist of the IP and the port and the answering message is, in case of an XML-RPC client, directly sent to him, or, in case of an AXIS client, put into the AXIS message cache.

```
public synchronized void notify(Object ob) {
  SensorValue sensorValue = (SensorValue) ob;
  //get the registry hashtable of the clients,
  //that registered for that sensor.
  Hashtable ht = (Hashtable) clientRegister.get(sensorValue.getSensorID());
  if (ht != null) {
    // if the hashtable isn't empty, iterate
    Iterator it = ht.keySet().iterator();
    while (it.hasNext()) {
      RegEntry temp = (RegEntry) ht.get((String) it.next());
      try {
        //make the call to clients
        makeCall(temp.getIp(), temp.getSensorID(), temp.getPort(), sensorValue)
    ;
      } catch (XmlRpcException e) {
        // if the client isn't reached for a reason, unregister him.
        unregister(temp.getIp(), temp.getSensorID());
      } catch (IOException e) {
        unregister(temp.getIp(), temp.getSensorID());
      }
    }
  }
  ht = (Hashtable) axisClientRegister.get(sensorValue.getSensorID());
  // get the registry hashtable of the clients, that registered for that sensor
    .
  if (ht != null) {
    Iterator it = ht.keySet().iterator();
    while (it.hasNext()) {
      //if the hashtable isn't empty, iterate
      RegEntry temp = (RegEntry) ht.get((String) it.next());
      Vector message = new Vector();
      //When the message cache already has an entry of that ip, add the message
     to it
      if (axisMessageCache.containsKey(temp.getIp())) {
        message = (Vector) axisMessageCache.get(temp.getIp());
        message.add(sensorValue);

        //when the amount of messages is higher than the threshold
        //and the database is initialized,than the messages are written to
        //the database
        if((message.size()>cacheSize)&&(useDB)){
          message.remove(0);
          writeToDataBase(temp.getIp(),message);
          message.clear();
          message.add(temp);
        }
        //if no entry exists, just add the SensorValue to the message
        } else {
          message.add(temp);
          message.add(sensorValue);
        }
        //everything is put in the AXIS message cache.
        axisMessageCache.put(temp.getIp(), message);
      }
    }
  }
}
```

The `makeCall` method is used for directly connection to XML-RPC clients. It's the key method in the notification mechanism

```
private synchronized void makeCall(String ip, String sensorID, String port,
        SensorValue sv) throws XmlRpcException, IOException {
  // A new XML-RPC client is initialized, containing the IP
  // and port of the registered client.
  xmlrpc = new XmlRpcClient("http://" + ip + ":" + port);
  Vector params = new Vector();
  String result = null;
  //adding the parameters to a Vector
  //to be conform with XML-RPC
  params.add(sv.getSensorID());
  params.add(sv.getDateStamp().toString());
  params.add(sv.getString());
  //a notify call is made to the "StableXMLRPCClient" handler
  result = (String) xmlrpc.execute("StableXMLRPCClient.notify", params);
}
```

If the database is used, the entries are saved in a local database everytime the user has more than 10 different messages. AXIS clients have actively to fetch the messages from the message cache by using the method `getAxisMessage`. Data that was saved in the database is fetched and removed from the database. As parameter, the IP of the client is needed. When the GatewayHandler is used separately from the XML-RPC Gateway, the `server.xmlrpc.localgateway` property in the `server.properties` file must be set at "true" The `GatewayHandler` runs with XML-RPC 1.1. The gateway handler has much the same methods as the XML-RPC gateway or the AXIS gateway, but it's supposed to be encapsulated from external access, and therefore not to be connected directly from qclients:

- `getAllLocations*()` : This method returns all location ids that sensors are located at.
- `getAllSensors*()` : This method returns all sensor ids no matter of their location.
- `getHardwareMetadata(String hardwareID)` : This method returns the hardware description of a sensor as Vector
- `getSensors*(String locationID)` : This method returns the sensor ids located at the specific location.
- `getSensorXML(String sensorID)` : This method returns the XML description of the sensor.
- `getServerDescription()` : This method returns the description of the server as String
- `getValue*(String sensorID)` : This method returns the last published value of that specific sensor.
- `getValues*(String sensorID)` : This method is for returning all values of a specific sensor.

- `getValuesXML(String sensorID, String startDate, String endDate)` : This method returns all events of a specific sensor from the starting date to the ending date.

- `notify(Object o)` : This method is used for client event notification.

- `register[Axis](String ip, String sensorID, String port)` : This method is called `registerAxis` for the AXIS gateway, and `register` for the XML-RPC gateway. It's for registering a client for asynchronous services and sensors.

- `unregister[Axis](String ip, String sensorID)`This method is called `unregisterAxis` for the AXIS gateway, and `unregister` for the XML-RPC gateway. It's for unregistering a client for a specific sensor or service.

- `useDB()` This method was implemented for the XML-RPC gateway and returns a boolean, if the database is running or not. It's called from the XML-RPC gateway to check if authentication is possible.

5. Sensors, Values, Locations and Adapter Modules

5.1. SensorHandler

This module handles the registration process for all sensors, sensor types, locations and hardware modules in the infrastructure. It also encapsulates the exploring methods for finding sensors via a specified sensor type, location, hardware group, etc. The `GatewayHandler` uses the methods of the `SensorHandler` to receive information about the registered sensors, locations and also to access the current or history sensor values. The `SensorPort` uses the register methods of the `SensorHandler` to pass the incoming registry requests (see figure 4.1).
All notifications of incoming sensor events through one of the gateway modules will be distributed to the required sensor in the private sensor hashtable.

```java
public boolean notifySensor(SensorValue val){
  if(hasSensorID(val.getSensorID())){
    ((Sensor)sensors.get(val.getSensorID())).notify(val);
    return true;
  } else return false;
}
```

The `SensorHandler` module can initialize the sensor types from the `sensortypes.xml` file located in the server root, as well as it parses the `locations.xml` and `sensors.xml` files to create the locations and sensors. You can see the sensor initialization as example in listing 5.1: If the property `server.initfromdatabase` is true (line 2), the `SensorHandler` will load the descriptions from the database (line 4). Then the module loads the path and name of the sensor description file (from the properties file) (line 7) and uses the `XMLProcessing` module to parse the XML file (line 10). Then it iterates through the parsed sensor list and adds the sensors to the private hashtable of sensors:

```java
private void initSensors(){
  if(preferences.getBooleanProperty("server.initfromdatabase")){
    if(useDatabase){
      this.readSensorsFromDatabase();
    }
  }
  String xmlfile = preferences.getProperty("server.sensorhandler.sensorfile");
  if(xmlfile != null) {
    String data = Utility.fileToString(xmlfile);
    ArrayList sens = XMLProcessing.parseSensorsXML(data);
    Iterator it = sens.iterator();
    while(it.hasNext()) {
      Sensor add = (Sensor)it.next();
      this.addSensor(add);
    }
  }
}
```

This class is implemented as singleton so there is no public constructor to prevent multiple instances. Use the `getInstance()` method to get access to the single instance of the `SensorHandler`.

5.2. Sensors

This class represents the server-side implementation of the connected sensors. The `SensorHandler` stores these sensor objects in the private hashtable and provides exploration methods.
Each sensor has a unique sensorID and also the properties that describes his measured values, the time intervalls, etc.
To export and import sensor information (e.g. used for the sensor registration process) the private member data of the sensor can be written to XML data, and also parsed from XML data back to the private members (by using the `toXML` and the `parseXML` method).
The `parseXML` method creates a new DOM `document` and creates a instance of the `SAXBuilder`.

```java
Document doc = new Document( );
SAXBuilder builder = new SAXBuilder();
```

Then the method tries to parse the contents of the `sensorXMLString` to the document `doc` (line 2), gets the root element (line 3) and validates that the root element is "Sensor" (line 4).

```
 1  try {
 2    doc = builder.build(new InputSource(new StringReader(sensorXMLString)));
 3    Element root = doc.getRootElement();
 4    if(root.getName().equals("Sensor")) {
```

In the next step the parser tries to read the two attributes of the `Sensor` root entry: the sensor class (line 1-2) and ths sensor id (line 4-5):

```
 1      Attribute attrib = root.getAttribute("class");
 2      if(attrib != null) sensorType = attrib.getValue().trim();
 3
 4      Attribute attrib2 = root.getAttribute("id");
 5      if(atttrib2 != null) sensorID = attrib2.getValue().trim();
```

To read the child nodes of the XML description, the method calls the `getChildren([name])` method of the root object (line 2). If there is an element in the `elements` list (line 3), the parser reads the text entry of the node (line 5):

```
 1      [...]
 2      List elements = null;
 3      elements = root.getChildren("Description");
 4      if(elements.size() > 0){
 5        Element descriptionElement  = (Element)(elements.get(0));
 6        description = descriptionElement.getTextTrim();
 7      }
 8      [...]
 9    }
10  }
```

The parsing process of the `Location` and `SensorType` objects works in a similar way.

5.3. Sensor Values

`SensorValue` objects contain the event messages of each sensor. When an event of a connected sensor occurs, the sensor adapter is responsible for passing this event as `SensorValue` to the platform (see section 5.5).
A `SensorValue` contains four main members:

1. `SensorID`: ID of the sensor that has produced the event message.

2. `Datestamp`: Date and time in ISO-8601 format (yyyy-MM-dd hh:mm:ss, e.g. 2005-12-30 22:12:17) [ISO 8601 Date and Time]

3. The value as string, float or integer. The members `valueString`, `valueFloat` and `valueInt` store the value.

4. `NativeType`: The native data type; set when you use the constructor or setter method. You can use the static constants of the `SensorValue` object to set or compare the native data type: `SensorValue.DATATYPE_INTEGER = 1`, `SensorValue.DATATYPE_FLOAT = 2`, `SensorValue.DATATYPE_STRING = 3`.

When you create a new `SensorValue` object, you can use one of the four constructor methods, e.g. the method `SensorValue(String sensorID, String value)` to specify the ID of the sensor and the event message. Each time when no datestamp is given to the `SensorValue` object, the current server date and time will be used for the date and time members.
The `SensorValue` can be written to XML data with the `toXML()` method (see listing 5.3): We use the JDOM library and create a new `Document` object (line 2). To the document we add the XML node object (see listing 5.3), format the XML output (line 4) and return the string contents of the XML document (line 5):

```
1 public String toXML(){
2   Document doc = new Document( );
3   doc.addContent(this.toXMLnode());
4   XMLOutputter out = new XMLOutputter(org.jdom.output.Format.getPrettyFormat())
      ;
5   return out.outputString(doc);
6 }
```

With the `toXMLnode()` method we get a XML child node with the contents of the `SensorValue`: We create a new XML `Element` and set the root entry to "Value" (line 2). Then we add the three members of each `SensorValue` (lines 3-5) and return the root object. The division in these two methods is because we use the `toXMLnode()` method also in the `XMLProcessing` class to generate sensor value collections.

```
1 public Element toXMLnode(){
2   Element root= new Element("Value");
3   root.addContent(new Element("SensorID").addContent(sensorID));
4   root.addContent(new Element("DateStamp").addContent(Utility.dateToString(
      dateStamp)));
5   root.addContent(new Element("Event").addContent(valueString));
```

```
  return root;
}
```

5.4. Locations

`Location` objects can specify the location of the sensor modules. Each object can contain the following members:

- `LocationID` (compulsory): ID string for the location.
- `LocationDescription` (compulsory): Description string for the location.
- `DegreeOfLongitude`: Degree of longitude as float.
- `DegreeOfLatitude`: Degree of latitude as float.
- `HeightAboveSeaLevel`: Height of the current location as float.
- `LocationType`: Integer value to specify if the location is inside or outside a building. Use the static final members of the `Location` object to specify: `LOCATION_NOT_SET = 0`, `LOCATION_INSIDE = 1`, `LOCATION_OUTSIDE = 2`.

Beyond the the getter and setter methods for the members, the `Location` object also has methods for reading and writing XML data: `parseXML()`, `toXML()` and `toXMLnode()`.

5.5. Notification

When new events of the connected sensors occured, the `SensorHandler` and especially the `Sensor` object that belong to the event have to be notified. Each sensor object implements the subject interface (`de.buw.medien.cscw.sensation.interfaces`), so that the `SensorHandler` can distribute the incoming sensor events to each sensor. The sensor receives the event as parameter (cast to `SensorValue`) and stores the events in a local cache (listing 5.5):

1. If `saveValuesInDatabase` is true (this is the private tag for the sensor to save values in database) and the `DataManager` is initialized, the sensor saves the value in the database (lines 2-10).

2. Then the sensor will iterate through the `observerList` and execute the notify method of each observer (this can be clients observing a sensor or a service object, etc.) (lines 12-15).

3. At last the incoming sensor value is stored in the local cache of sensor values (by default 30 values) (line 16).

```
1  public void notify(Object event) {
2    if(saveValuesInDatabase){
3      DataManager dbm = sensorHandler.getDatabaseManager();
4      if(dbm != null){
5        SensorValue sv = (SensorValue)event;
6        sv.setNativeType(this.nativeDataType);
7        ArrayList events = new ArrayList();
8        events.add(sv);
9        dbm.insertSensorValues(events);
10     }
11   }
12   Iterator it = observerList.iterator();
13   while(it.hasNext()){
14     ((ObserverInterface)it.next()).notify(event);
15   }
16   this.addValueToCache((SensorValue)event);
17 }
```

To register a new observer for the sensor object you have to execute the `register` method and add the observer object as parameter (`public boolean register(Object observer)`). The observer must implement the observer pattern (`de.buw.medien.cscw.sensation.interfaces`), so that he has the notify method to handle the sensor events (to unregister use the method `public boolean unregister(Object observer)`).

5.6. Sensor Types

The sensor types specify the class affiliation of a sensor, e.g. as temperature or movement sensor. These are classifications to list sensors with the similar measurements together in the same group. The sensor types are specified in the `sensortypes.xml` file, and the `SensorHandler` reads this file and creates the

`SensorType` objects. Each registered `Sensor` has to refer to a sensor type. The following sensor types are specified in the `sensortypes.xml`:

ESB Sensors	
Temperature	Temperature sensor
Movement	Movement detection (with passive IR)
Vibration	Vibration detection
Light	Light sensor
Noise	Current microphone level
NoiseAverage	Average microphone level
NoiseCounter	Counter value of the microphone
Button	Hardware button of the ESB
Infrared	Infrared remote control
Mobile Sensors	
CellPhoneText	Text messages from mobile phone
CellPhoneState	State description of mobile phone
CellPhoneRC	Remote control command
Other Sensors	
Presence	Presence information of the PRIMI messenger
ASCII	General text message
MessengerStatus	Messenger state in general
MessengerText	Mesenger text message
Binary	True or false state of a sensor
Service	A sensor created by a service object
Other	None of the categories above

5.7. XML Descriptions

The sensors can be described in XML format, to enable the easy exchange and import of sensor information. Use the `toXML` method of a sensor object to export to an XML string (as in listing 5.1) or use the `parseXML` method to parse a string and insert the values into the sensor objects members.

Listing 5.1: Sensor XML description

```xml
<Sensors>

  <Sensor id="SensorESBButton" class="Button">
    <Description>ESB hardware button, true or false.</Description>
    <LocationID>HK7</LocationID>
    <Owner>Cooperative Media Lab</Owner>
    <Comment>Binary hardware button control</Comment>
    <AvailableSince>2005-01-01 10:00:00</AvailableSince>
    <AvailableUntil>2005-12-01 12:00:00</AvailableUntil>
    <SensorActivity activity="active" />
    <NativeDataType>Binary</NativeDataType>
    <MaximumValue></MaximumValue>
    <MinimumValue></MinimumValue>
    <HardwareID></HardwareID>
    <Command></Command>
  </Sensor>

  <Sensor id="SensorESBTemp" class="Temperature">
    <Description>Embedded sensor board temperature sensor.</Description>
    <LocationID>HK7</LocationID>
    <Owner>Cooperative Media Lab</Owner>
    <Comment>Measures temperature in celsius degree.</Comment>
    <AvailableSince>2005-01-01 10:00:00</AvailableSince>
    <AvailableUntil>2005-12-01 12:00:00</AvailableUntil>
    <SensorActivity activity="active"></SensorActivity>
    <NativeDataType>Float</NativeDataType>
    <MaximumValue>50.0</MaximumValue>
    <MinimumValue>-40.0</MinimumValue>
    <Unit>Celsius</Unit>
    <HardwareID />
    <Command />
  </Sensor>

  [....]

</Sensors>
```

5.8. Integrate new Sensors

To integrate new sensors into the platform you can:

1. Add the XML description to the `sensors.xml`, so the sensor will be initialized at startup.

2. Use the PHP adminstration interface to register the new sensor (see 8.2).

3. Use the XML-RPC interface to register the new sensor via the `SensorPort` (contact the running server with IP (`localhost` if the server is running on your local computer) and the port (by default 5000)).

In chapter 10.2 you can find a tutorial for the development of a sensor adapter class (a virtual keyboard sensor).

5.9. Adapter Modules

The adapter modules are the connector between the sensors hardware components and the server side. Each sensor adapter can choose one of the notification gateways to communicate with the server, for example the XML-RPC interface `SensorPort`, the socket interface `GatewaySocket` or the HTML lightweight gateway.
Adapters hide the sensor hardware specific implementation (e.g. complex terminal commands) and are responsible for the data retreival of each sensor. With one adapter module it is possible to connect more than one sensor to the platform; in that case the adapter must assign the correct sensorID to the published events. Each time when an event occurs, the adapter is responsible for the transmission to the server.
Adapters can connect various hardware modules to the server, e.g. connected at the serial (COM) or parallel port (LPT) as well as USB or Firewire port.

5.9.1. ESB XML-RPC Adapter

The ESB (Embedded Sensor Board, see [ESB Documentation]) is a hardware module with some integrated sensors, e.g. the sensors for temperature, movement (passive infrared detection), light intensity, vibration or noise (as well as some further sensors). The ESB is connected to the COM port of the computer, so we used this module connected to the Windows PC (due to the missing COM port of the iMacs).
For the connection to the ESB board we have developed several classes to encapsulate the complete hardware connection and hide this implementation from the server side. You find these communication classes in the package `de.buw.medien.cscw.sensation.sensors`, the test classes in `de.buw.medien.cscw.sensation.sensors.tests` and the adapter in `de.buw.medien.cscw.sensation.sensors.adapter`.

5.9.2. ESB Communication Parser

One of the important classes of the ESB adapter is the `SensorParser` that can translate the received and coded messages of the ESB board. The `SensorParser` parses the string that contains summarized information from the ESB sensor board

and extracts the sensor values to add them to the `SensorDataCollection` object. The ESB terminal command [ESB Terminal Commands] for receiving all sensor values is the "rsr" command. With the "sft 32" flag you can activate the automatic sending of all current sensor events and values. This also includes remote events, transmitted from wireless connected further ESB modules using the integrated transceive/receive hardware.
Here is an example of an received string of the ESB:

```
[20|01.01.00 00:04:46|+026.0][IR: C(5) A(20)][Btn: 0][Light: 1046 Hz][Pir: 2][Vib: 0][Mic: 0][BAT: 2266][EXT: 158]
```

To parse this string we use the regular expression package [Darwin 2001] of the java utilility classes and an instance of the `Pattern` and `Matcher` objects:

```
import java.util.regex.*;
[...]
  Pattern pattern;
  Matcher matcher;
```

Then we initialize the `pattern` with an object of the static compile method of the `Pattern` class, and we specify the regular expression pattern (for finding the string "[Light: 1046 Hz]") as parameter (line 1). The `matcher` object gets an instance created by the `matcher` method of the `pattern` object and gets the string to parse as parameter. If the pattern was found, the parser extracts the numeric value with a private method (line 5) and sets the value of the light member of the `SensorDataCollection` to the integer value of the parsed string (line 7).

```
1    pattern = Pattern.compile("\\[Light:\\s[0-9]*\\sHz\\]");
2    matcher = pattern.matcher(toParse);
3    if (matcher.find()){
4      String light = matcher.group();
5      light = this.regularExpressionNumbers(light);
6      if (!light.equals("")){
7        collection.setLight(this.stringToInt(light));
8      }
9    }
```

The parsing process for the further data members of the value string is similar to this explained example. See the Javadoc notes in the `SensorParser.java` file for more information.

6. Services

6.1. Concept of Services

With the `SensorHandler` (chapter 5) and the gateway modules (chapter 8) the client can explore the registered sensors and can also access their values. To further process this raw data, the client has to implement these calculations in his own methods. To transfer this processing step to the server methods, we have implemented the *Service* classes.
The services can interpret sensor values, but also aggregate values of different sensors. They make it possible to create a sort of logic network of connections between sensors. In example a service can gather the values of all registered temperature values and calculate the average temperature value. Or the service can observe all movement sensors and reacts if one of the sensors detects movement (see figure 6.1).

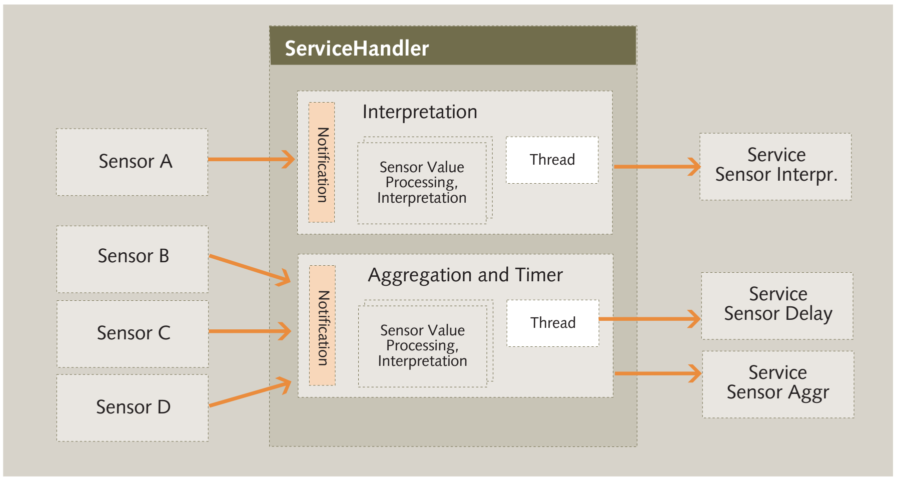


Figure 6.1.: The service architecture: Interpretation and aggregation of sensor values

With the implementation of `Service` classes we can build objects which are able to build the functionality of *transformers*, *filters*, *mergers* and *aggregators* (as explained in [Chen & Kotz 2002]). To implement a *transformer* (we call this *interpreter*) you have to register the service for exact one sensor and implement the interpretation functionality in the `notification()` method. The *merger* subscribes for more than one sensors (of the same sensor type) and calculates for example the average value (of temperature sensors). The *filter* will for example only pass the values above a given threshold and the *aggregator* can build interpretations on the basis of different kind of sensors (see figure 6.2).

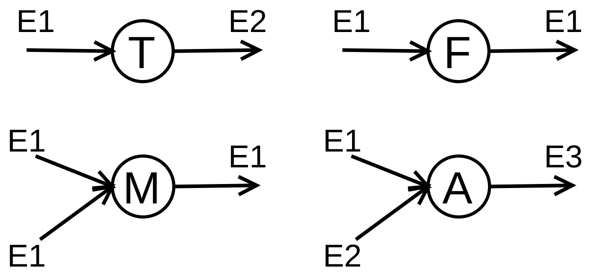


Figure 6.2.: Four types of operators [Chen & Kotz 2002]

Chen and Kots [Chen & Kotz 2002] also explain the different connections between the single operators (in *Sens-ation*: *services*). Figure 6.3 illustrates the structures for connections of the infrastructure and the applications (white circles are information sources/sensors, white squares are operators/services and dark rectangles represent the application-specific processing [Chen & Kotz 2002]):

a) Infrastructure sends raw data from the sensors to the application, so the application has to interpret these data.

b) The service of the infrastructure provides high-level information, but some application-specific processing is still necessary.

c) The application-specific processing is implemented in the infrastructure network.

d) The processing of sensor data is decomposed in application-independent portions and application-specific portions.

e) More complex network of connections between the services (and applications can access different connections points of this network).

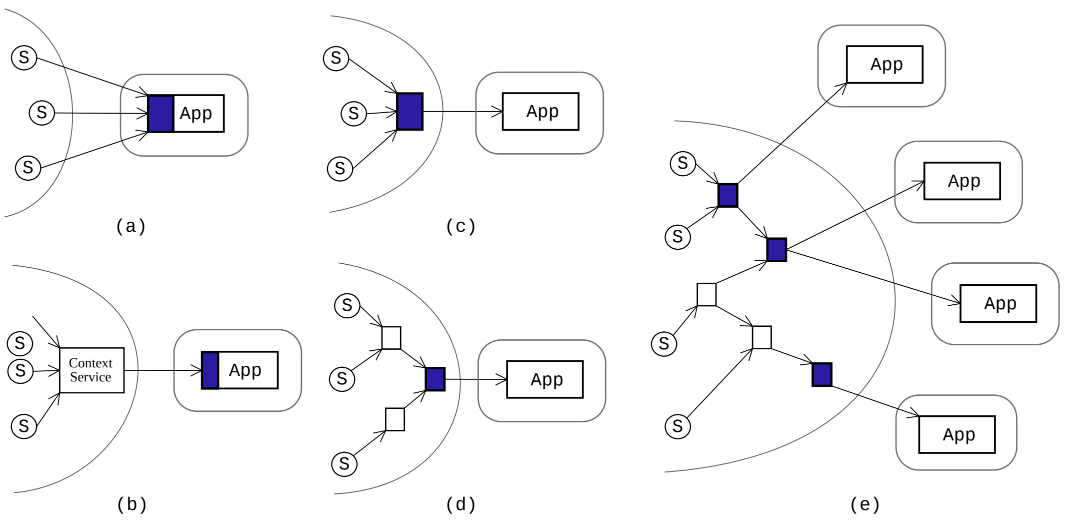


Figure 6.3.: Methods for sensor data interpretation [Chen & Kotz 2002]

These structures can be implemented with using the `Service` classes of the *Sensation* platform (section 6.3 describes the development of a new `Service` class file). With these operators (*services*) and connections we can build more complex networks to process the sensor data and retrieve context information or other high-level interpretations. Figure 6.4 illustrates the composition of a operator graph and show the flow of raw source events through the interpretation process.

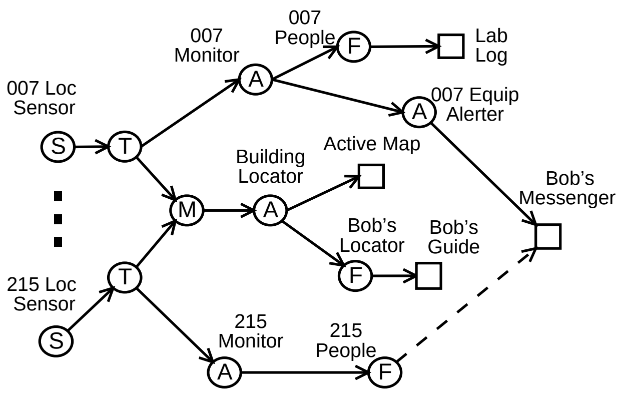


Figure 6.4.: Example operator graph [Chen & Kotz 2002]

6.2. ServiceHandler Module

By analogy with the `SensorHandler`, the `ServiceHandler` has `register` and `unregister` methods for services. The `initServices` method uses the dynamic class loader `URLClassLoader` to create instances of all specified services in the 'services.xml' file (listing 6.1). This technology can be extended to a plug-in technology, e.g. to load service classes from remote locations.

Listing 6.1: ServiceHandler dynamic class loading (without try/catch blocks)

```
String data = Utility.fileToString(xmlfile);
ArrayList serviceLoad = XMLProcessing.parseServiceLoadXML(data);

Iterator it = serviceLoad.iterator();
while(it.hasNext()) {
  String pathToClass = (String)it.next();

  URL url = new File("").toURL();
  URLClassLoader cl = new URLClassLoader( new URL[]{ url } );

  Class c = cl.loadClass(pathToClass);
  this.registerService((Service)c.newInstance());
}
```

Two further methods are responsible for passing register and unregister notifications to the services: the methods `notifySensorRegister` and `notifySensorUnregister`. The `SensorHandler` sends a notification to the `ServiceHandler`, and the latter passes these infromation to all registered sensors. They implement these notification methods from the abstract `Service` class. So each service has to decide how to react to this register/unregister information.

6.3. Development of Services in General

To create a new service for the platform, you can follow these guidelines for service development:

1. Create a new class in the package *de.buw.medien.cscw.sensation.server.services*.

2. Derive your class from the abstract service class: add "extends Service".

3. Create your class constructor, add 'super("serviceID")' to the constructor (where serviceID is the name of your service).

4. Add a new sensor

5. Register your service as observer for all sensors you are interested in.

6. Implement 'notify' method: you receive values from all sensors you are registered for as observer. In the notify method you can handle these events.

7. Implement 'run' method: if you want use timer or delay functions

8. Implement 'notifySensorRegister' and 'notifySensorUnregister' methods: The SensorHandler sends you an information, when a sensor registers or unregisters himself from the infrastructure, so you can decide if your service has to suspend until the sensor registers again.

9. Add your class name to the 'services.xml' file. Add

   ```
   <ServiceClass>
     de.buw.medien.cscw.sensation.server.services.[yourClassName]
   </ServiceClass>
   ```

   to the `ServiceLoad` section.

10. Test the service in the development framework: with the build.xml file you can compile and run the server framework; your new class will be compiled as well. With "list services" you can list the services to the server console.

6.4. Example Service: Interpreter

To illustrate the necessary development steps to create a new service, we will explain the following example of an interpreter service.

1. Create the new class file "Interpreter.java" in the package *de.buw.medien.cscw.sensation.server.services.*

2. Add "extends Service" to the class initialization (and if you use Eclipse [Eclipse Foundation] you can automatic implement the necessary methods of the super class).

```
public class ServiceExample extends Service {
}
```

3. Create your class constructor and add 'super("Interpreter")' to the constructor. Create a call of the `registerSensor()` method we create in the next step.

```
    public Interpreter() {
        super("Interpreter");
        this.registerSensor();
    }
```

4. Implement the `registerSensor()` method:

```
private boolean registerSensor(){
  String regSensorString =  "<Sensor id=\"InterpreterSensor\" class=\"
    Service\">" +
    "<Description>Our new temperature interpreter</Description>" +
    "<HardwareID></HardwareID>" +
    "<Command></Command><LocationID>HK7</LocationID>" +
    "<Owner>You</Owner><Comment>Temp. Interpreter</Comment>" +
    "<AvailableSince>2005-01-01 10:00:00</AvailableSince>" +
    "<AvailableUntil>2005-12-01 12:00:00</AvailableUntil>" +
    "<SensorActivity activity=\"active\" />" +
    "<NativeDataType>String</NativeDataType>" +
    "<MaximumValue></MaximumValue>" +
    "<MinimumValue></MinimumValue>" +
    "</Sensor>";

  Sensor regSensor = new Sensor();
  notificationSensorID = regSensor.parseXML(regSensorString, true);
  return sensorHandler.addSensor(regSensor);
}
```

5. Register your service as observer for one existing sensor in the infrastructure; here: the sensor "ESB1Temp" (the ESB temperature sensor). Add the following statement to the constructor:

```
sensorHandler.register("ESB1Temp", this);
```

6. Now we implement 'notify' method: we cast the notification object to a sensor value and decide how we have to interpret the current temperature:

```
public void notify(Object event) {
  SensorValue sv = (SensorValue) event;
  Float temperature = event.getFloat();
  SensorValue notificationValue = new SensorValue();

  if(temperature > 30.0f){
    notificationValue.setValue("It is hot!");
  } else if(temperature < 5.0f){
    notificationValue.setValue("It is cold!");
  } else {
    notificationValue.setValue("Normal temperature.");
  }
  sensorHandler.notifySensor(notificationValue);
}
```

7. We only implement the 'notifySensorRegister' and not the 'notifySensorUnregister' method: so we will be notified when our observed sensor is registered again to the platform. We ignore the case that the sensor is not available anymore.

```
public void notifySensorRegister(Sensor sensor) {
  if(sensor.getSensorID().equals("ESB1Temp")){
    sensorHandler.register("ESB1Temp", this);
  }
}
```

8. Add the class name to the 'services.xml' file. Add

   ```
   <ServiceClass>
     de.buw.medien.cscw.sensation.server.services.Interpreter
   </ServiceClass>
   ```

   to the `ServiceLoad` section.

9. Test the service in the development framework: with the `build.xml` file you can compile and run the server framework; your new class will be compiled as well. With "list services" you can list the services to the server console.

7. Database

The Sens-ation infrastructure captures information about sensors, their locations and their measured values. To make this data available for longer time periods we need a database. We decided to use the open source database MySQL [MySQL Database Website]. The Java Database Connectivity (JDBC) is used to access the database. This chapter describes the database structure and the Sens-ation package `de.buw.medien.cscw.sensation.database`.

7.1. Using JDBC

JDBC is a Java API (included in J2SE and J2EE) to provide connectivity to SQL databases [JDBC]. To access the MySQL database JDBC needs a MySQL driver. We are using MySQL Connector/J [MySQL Connector/J].

After installing MySQL Connector/J by simply adding the JAR file to the class-path the implementation is done in a few steps:

1. Load the JDBC-driver (see line 13 in listing 7.1)
2. Establish the connection (see line 15 in listing 7.1)
3. Generate a SQL-statement (see line 19 in listing 7.1)
4. Execute the SQL-statement (see line 21 in listing 7.1)
5. Get the result (`ResultSet` line 21 in listing 7.1)
6. Close the connection (see line 29 in listing 7.1)

A simple example:

Listing 7.1: Using JDBC

```
1  // Set the variables
2  // Driver name
3  String driver = "com.mysql.jdbc.Driver";
4  // Url of the database
5  String url = "jdbc:mysql://" + database.hostIP + "/" + database.name;
6  // User name
7  String user = "test";
8  // User password
9  String password = "test";
10
11 try {
12   // Load the driver
13   Class.forName(driver).newInstance();
14   // Establish the connection
15   Connection con = DriverManager.getConnection(url, user, password);
16   // Create the statement
17   Statement stmt = con.createStatement();
18   // Create the SQL query
19   String sqlQuery = "SELECT * FROM " + tableName;
20   // Execute the query
21   ResultSet rSet = stmt.executeQuery(sqlQuery);
22
23   // Get the result
24   while (rSet.next())
25     System.out.println (rSet.getString(1));
26
27   // Close
28   stmt.close();
29   con.close();
30
31 } catch (Exception e) {
32   System.out.println("DB: error could not load driver");
33 } catch (SQLException e) {
34   System.out.println("DB: SQLException: " + e.getMessage() +
35          ", SQLState: " + e.getSQLState() +
36          ", VendorError: " + e.getErrorCode());
37 }
```

The `java.sql.ResultSet` object (see line 21 in listing 7.1) is the return type of a JDBC SQL request. The result of the request is given as a table.

7.2. Database Structure

The Sens-ation framework creates the following tables:

- **Location table**

The location table contains all registered locations (section 5.4). The locationID is used as key.

locationID	description	longitude	latitude	heightAboveSealevel	locationType

Figure 7.1.: Location table

- **Sensor table**
  The sensor table contains all registered sensors. The sensorID is used as primary key. The locationID references to the location table.

sensorID	hardwareID	command	sensorType	description	nativeDataType	maximumValue	minimumValue	locationID	pollingInterval	owner	comment	availableSince	availableUntil	isRemoteSensor	isPublishingSensor

Figure 7.2.: Sensor table

- **Sensor history table**
  This table has the same structure as the sensor table (see above), but contains all deleted sensors.

- **Sensor values table**
  The sensor value table contains all sensor values [1]. The sensorID references to the sensor table.

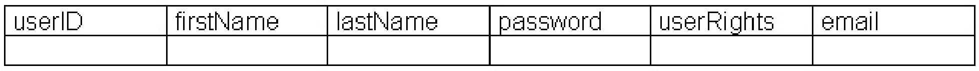
starttime	endtime	sensorID	nativeType	intValue	floatValue	stringValue	count

Figure 7.3.: Sensor value table

- **Sensor values history table**
  This table has the same structure as the sensor value table (see above), but contains all deleted sensor values.

[1]Null events are not saved into the database. See section 7.3.2

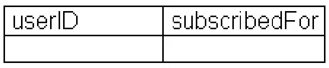
- **User table**
  The user table contains all registered users. The userID is used as key.



userID	firstName	lastName	password	userRights	email

Figure 7.4.: User table

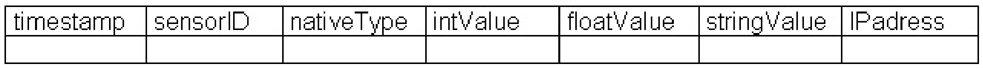
- **User subscribe table**
  This table contains all sensors and services a user is registered for.



userID	subscribedFor

Figure 7.5.: User subscribe table

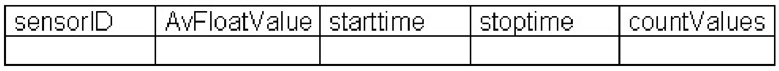
- **Axis event table**
  The Axis event table contains Axis events as cache for the Axis server, because the Axis server is not able to notify the Axis clients. So this table stores the events until the client requests for them. It contains the IP addresses corresponding to the events. An event consists of a sensor value (section 5.3).



timestamp	sensorID	nativeType	intValue	floatValue	stringValue	IPadress

Figure 7.6.: Axis event table

- **Average sensor value table**
  This table contains the average sensor values of all sensors.



sensorID	AvFloatValue	starttime	stoptime	countValues

Figure 7.7.: Average sensor value table

7.3. Implementation

7.3.1. Database Package

The classes of the package `de.buw.medien.cscw.sensation.database` are for encapsulation the database access. For using the classes of this package you does not need any knowledge about SQL and JDBC.

The class `Database` is an abstract super class. The constructor of these super class prepares the database access. It loads the driver, establishes and holds the connection.
The super class includes also these methods:

- Execute a SQL query
- Create a table
- Get database entries from a table
- Delete a table
- Convert the `ResultSet` into an Java object of the Sens-ation framework (subsection 7.1)

For handling the Java objects of the Sens-ation framework there are the following extended classes:

- `AxisEventDatabase`:
  Handles axis events (section 4.4)
- `DataManager`:
  Handles locations (section 5.4), sensors (section 5.2) and sensor values (section 5.3)
  Implements the `DataManagerInterface`

- `ServiceDatabase`:
  Handles average sensor values
  Implements the `ServiceDatabaseInterface`

- `UserDatabase`:
  Handles the registered users

The class `SensorValueDBEntry` is the type for saving sensor values in the database. So a `SensorValue` must be converted before it will be added to the database (section 7.3.2).

7.3.2. Special Methods

The methods for

- Creating a table
- Updating a table
- Inserting entries
- Converting entries

are attuned to the corresponding classes. The creating table methods have hard coded strings for the table columns. Cause the tables have to save all the members from the corresponding class and these are fixed.
So if you changes such a class, for example you delete or include new members in the class `Location` you have to change the `SQLquery` string in `createTableLocation()`, `insertLocations()`, `updateLocationTable()` and in the class `Database` the result set parsing (section 7.1) of `convertIntoLocationArrayList()`.

Insertion of Sensor Values

The method `insertSensorValues()` is a method of the `DataManager`. This method stores the sensor values in the database. The `insertSensorValues()`

method checks the native type of the sensor values and, if it is a number, the method calls `insertSensorValueIntoHashtable()`.

The method `insertSensorValueIntoHashtable()` includes an algorithm for data compression. These algorithm affects only sensor values, which have a number as native type[2]. For the compression the method stores temporarily the sensor values in the hashtable `sensorValue`.

The behavior of `insertSensorValueIntoHashtable()`:

Listing 7.2: Insertion sensor values into the hashtable

```
if(sensorID of the sensor value not in hashtable){
  add the sensor value to the hashtable
  if(sensor value == 0){
    insert an 'on-stamp' into the database
  }
}
else{
  if(hashtable entry of sensorID == sensor value){
    set the stoptime of the hashtable entry to the timestamp of the sensor
    value
  }
  else{
    if (hashtable entry of sensorID != 0){
      insert the hashtable entry into the database
    }
    add the sensor value to the hashtable
  }
}
```

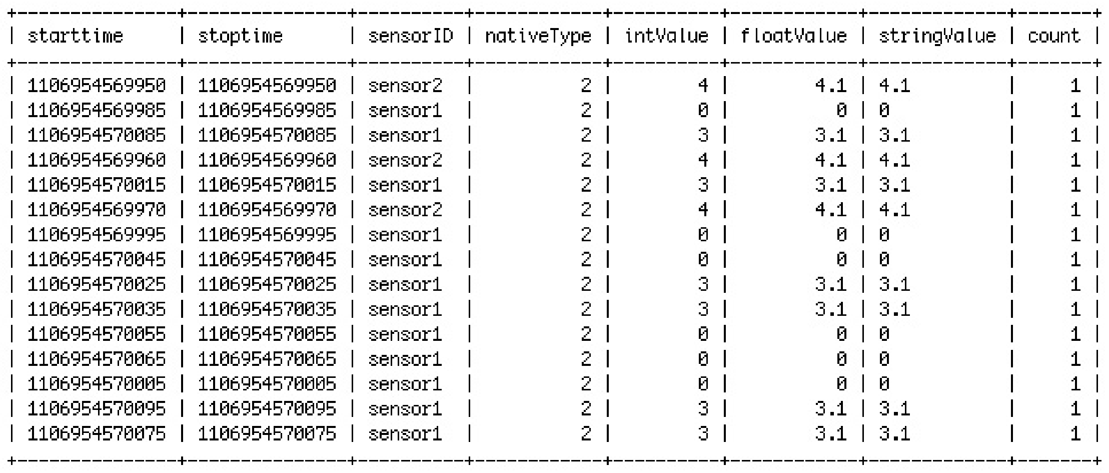
An uncompressed table is shown in figure 7.8.

In the table the start time and the stop time are saved as `Java.lang.Long` type. That allows to save the datestamp of the sensor values precise up till the millisecond. The time data types of the MySQL database do not provide saving time such accurately.

Figure 7.8 shows that the 'sensor1' sent: *0*, *0*, *0*, *3.1*, *3.1*, *3.1*, *0*, *0*, *0*, *3.1*, *3.1* and *3.1*. The 'sensor2' sent only three times the value *4.1*. Figure 7.9 shows the table of figure 7.8 after compression.

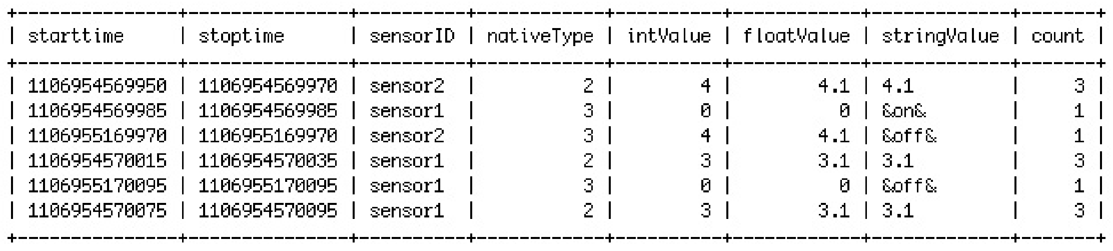
In figure 7.9 shows that no 'null events' are in the table. The first entry of the 'sensor1' is an 'on-stamp'. The three values *3.1* are compressed to one entry. The start time is the timestamp of the first value, the stop time is the timestamp of the

[2]Sensor Values, that are strings, are stored uncompressed in the database by the method `insertSensorValues()`.



```
+---------------+---------------+----------+------------+----------+------------+-------------+-------+
| starttime     | stoptime      | sensorID | nativeType | intValue | floatValue | stringValue | count |
+---------------+---------------+----------+------------+----------+------------+-------------+-------+
| 1106954569950 | 1106954569950 | sensor2  |          2 |        4 |        4.1 | 4.1         |     1 |
| 1106954569985 | 1106954569985 | sensor1  |          2 |        0 |          0 | 0           |     1 |
| 1106954570085 | 1106954570085 | sensor1  |          2 |        3 |        3.1 | 3.1         |     1 |
| 1106954569960 | 1106954569960 | sensor2  |          2 |        4 |        4.1 | 4.1         |     1 |
| 1106954570015 | 1106954570015 | sensor1  |          2 |        3 |        3.1 | 3.1         |     1 |
| 1106954569970 | 1106954569970 | sensor2  |          2 |        4 |        4.1 | 4.1         |     1 |
| 1106954569995 | 1106954569995 | sensor1  |          2 |        0 |          0 | 0           |     1 |
| 1106954570045 | 1106954570045 | sensor1  |          2 |        0 |          0 | 0           |     1 |
| 1106954570025 | 1106954570025 | sensor1  |          2 |        3 |        3.1 | 3.1         |     1 |
| 1106954570035 | 1106954570035 | sensor1  |          2 |        3 |        3.1 | 3.1         |     1 |
| 1106954570055 | 1106954570055 | sensor1  |          2 |        0 |          0 | 0           |     1 |
| 1106954570065 | 1106954570065 | sensor1  |          2 |        0 |          0 | 0           |     1 |
| 1106954570005 | 1106954570005 | sensor1  |          2 |        0 |          0 | 0           |     1 |
| 1106954570095 | 1106954570095 | sensor1  |          2 |        3 |        3.1 | 3.1         |     1 |
| 1106954570075 | 1106954570075 | sensor1  |          2 |        3 |        3.1 | 3.1         |     1 |
+---------------+---------------+----------+------------+----------+------------+-------------+-------+
```

Figure 7.8.: Uncompressed database table



```
+---------------+---------------+----------+------------+----------+------------+-------------+-------+
| starttime     | stoptime      | sensorID | nativeType | intValue | floatValue | stringValue | count |
+---------------+---------------+----------+------------+----------+------------+-------------+-------+
| 1106954569950 | 1106954569970 | sensor2  |          2 |        4 |        4.1 | 4.1         |     3 |
| 1106954569985 | 1106954569985 | sensor1  |          3 |        0 |          0 | &on&        |     1 |
| 1106955169970 | 1106955169970 | sensor2  |          3 |        4 |        4.1 | &off&       |     1 |
| 1106954570015 | 1106954570035 | sensor1  |          2 |        3 |        3.1 | 3.1         |     3 |
| 1106955170095 | 1106955170095 | sensor1  |          3 |        0 |          0 | &off&       |     1 |
| 1106954570075 | 1106954570095 | sensor1  |          2 |        3 |        3.1 | 3.1         |     3 |
+---------------+---------------+----------+------------+----------+------------+-------------+-------+
```

Figure 7.9.: Compressed database table

third value and the counter is three. For the 'sensor2' there are only two entries. The first three values are compressed to one.

Saving no 'null events' generates a problem. If the table contains no events for requested time you does not know, whether the sensor sent really 'null events' or the sensor was off. So every time sensor events are sent, `insertSensorValues()` calls `checkHashtable()` (see section 7.3.2) which sets an 'off-stamp', if a sensor value is older than 10 minutes.

Checking the Hashtable

The compressing of the data generates the problem, if the database contains no sensor values, one does not know, if the sensor was unreachable or the sensor measured are 'nulls'. So one has to check and save whether a sensor works. This

is done by the method `checkHashtable()`.
Every time new values from the sensors are received, the method is called to check the entries in the hashtable. If there are values, which are older than 10 minutes, they were inserted to the database, also an 'off-stamp' were set in the database and the hashtable entries were deleted. The sensors does not send any values.

Clearing the Local Cache

The local cache is a hashtable called `sensorValues` and is used for storing sensor values temporarily before compression (section 7.3.2). This hashtable contains at least the last sensor value of a sensor. While shut down the server the entries of this hashtable have to be inserted into the database, otherwise they are lost. Therefore the method `clearLocalCache()` is called.

Reconstructing Sensor Values

Because the database contains compressed sensor values you need of course a method to decompress these entries. This is done by `reconstructSensorValues()`.
With `getSensorValues()` you get all stored sensor values of a specified *sensorID* from the database. Optionally you can specify a time interval. So this method returns only values, which are measured in this interval. If the native type of the sensor values is a number, the `reconstructSensorValues()` method is called. Those needs as parameters an `ArrayList` of the selected entries, the sensor value before the first selected one, the start time and the end time given by the user.

The `reconstructSensorValues()` method processes tho following steps:

1. Calculate the time interval between two sensor values.

2. Compare the start time (specified by the user) with the timestamp of the first entry of the `ArrayList`. Only if the start time is earlier than the first timestamp and the sensor value before is not an 'off-stamp', 'null events' are generated.

3. Decompress the entries of the `ArrayList` while verifying the following to conditions:

  - If the count of the current sensor value is one, generate one sensor value, else the number of generated values as much as the count.
  - If the timestamp difference between the sensor values is greater than the calculated time interval (see step one), 'null events' are generated.

4. Compare the timestamp of the last sensor value from the `ArrayList` with the end time (specified by the user). Only if the following three conditions are true 'null events' are generated.
   - The end time is later than the last timestamp.
   - The difference between end time and last timestamp is greater than the time interval of two sensor values.
   - The sensor value before is not an 'off-stamp'.
5. Return an `ArrayList` of decompressed sensor values.

'Null events' are generated by the method `generateNullValues()`. These needs as parameters a start time and a stop time. Two sensor values with 'null' as value are created. Both are getting the corresponding timestamp (start time respectively stop time).

Erasing Sensors

We are using the standard table type 'MyISAM' of the MySQL database. It does not support foreign keys. While creating a table the MySQL server parse for foreign keys, but does not save this information [MySQL and Foreign Keys]. So it is necessary to develop methods for preserve the referential integrity.
Such a method is `deleteSensor()`. The 'sensorID' in the sensor value table references the sensor table. If you delete a sensor from the sensor table, the corresponding sensor values are still in the sensor value table. So there are sensor values of an unknown sensor. This values are unusable for the Sens-ation framework.

Therefore the `deleteSensor()` method does not only delete the sensor but also:

- Creates a sensor history table and writes the deleted sensor into this one.

- Creates a history table for the sensor values and displaces all values from the deleted sensor into it.

Consequently the referential integrity is still preserved.
A final erasing of sensors or sensor values is not implemented yet. If you want to do this even though, you have to use the MySQL client.

8. Gateways

8.1. Web Services

8.1.1. XML-RPC

E**X**tended **M**arkup **L**anguage - **R**emote **P**rocedure **C**all (XML-RPC) is a specification that is constructed for resolving communication between different technical and operational environments. It uses HTTP as transport protocol as well as xml (Extended Markup Language) for formatting data. XmlRpc can instanciate Webservers and their Webclients to resolve communication. A typical call looks like that.

```

POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181
<?xml version="1.0"?>

<methodCall>
  <methodName>GatewayXMLRPC.getValueVector</methodName>
   <params>
      <param>
         <value><string>Sensor1ESBTemp</string></value>
         </param>
      </params>
</methodCall>
```

This excerp shows, that the data consists of a method call that tries to start the method `getValueVector` on the handler `GatewayXMLRPC` with a parameter of datatype `String` consisting of `SensorESBTemp`. The answer of the call would be much like the same but only consisting of values that are to be returned.

The implementation of the XML-RPC client side in Java is done as followed:

```java
private XmlRpcClient xmlrpc;

try {
      xmlrpc = new XmlRpcClient("http://141.54.159.129:5000");
    } catch (MalformedURLException e) {
      System.out.println("Error initializing XmlRpcClient" );
    }
```

This code shows the Java implementation on the XML-RPC server:

```java
try {
        server = new WebServer(5000);
    } catch (IOException e1) {
        Logger.message("ERROR: Can't start XML-RPC service. Please check if
    port 5000 is available.");
    }
```

An implementation of a method in Java that starts an XML-RPC call looks like that:

```java
public synchronized String getSensorXML(String sensorID) {
    String ret = null;
    Vector params = new Vector();
    params.add(sensorID);
      try {
        ret = (String) xmlrpc.execute("GatewayHandler.getSensorXML", params);
      } catch (XmlRpcException e) {
        System.out.println("GatewayXMLRPC : Error in getting Sensor as XML");
      } catch (IOException e) {
        System.out.println("GatewayXMLRPC : Error in getting Sensor as XML");
      }
    System.out.println("XMLRPC-Method called: getSensorsXML(sensorID)");
    return ret;
  }
```

The method shows how XML-RPC calls are made. The `XmlRpcClient xmlrpc` has a public method `execute` which uses as parameters the handler's name, by that the method that is registered, and the methods name itself. The parameters for the called method are added in a Vector and added to the execute call.

8.1.2. Gateway XML-RPC

The GatewayXMLRPC class is, as the name tells, the gateway for XML-RPC calls. It is automatically installed with the server package (see 3.2.2) It is a XML-RPC client to the `GatewayHandler` class and has its own webserver, such enabling not only to send, but also to receive calls from the handler. It can be run separately or together with the gateway handler by setting the `localgateway` property in `gatewayxmlrpc.properties` file to 0 or 1. It features a security mechanism for DoS attacks, that blocks calls to the gateway handler, reducing the amount of traffic that is passing through the server network. That is done by getting the time difference in milliseconds from one request to the next, and comparing it to a threshold that is determined in the `gatewayxmlrpc.properties` file named `timethreshold`. If the timelag between the requests is to small, already collected data from the cache is returned.

```java
private synchronized SensorValue checkCritTime(String sensorID) {
  SensorValue sv = new SensorValue();
  //if the sensorValue cache is not empty
  if (cache != null) {
    //when the cache has an entry for that sensor
    if (cache.containsKey(sensorID)) {
      //get the entry
      PEntry pe = (PEntry) cache.get(sensorID);
      if (pe != null) {
        long savedMillis = pe.itsMillis;
        long now = new Date().getTime();
        //when the time difference is smaller as the
        //threshold, return a filled value, else an empty one
        if ((now - savedMillis) < timethreshold) {
          sv = pe.itsSv;
  } } } }
  return sv;
}
```

In the methods using this feature (namely `getValueVector`, `getValueHashtable` and `getValueHashMap`) check if the value that is returned from the method is empty. If it is, the request is forwarded to the gateway handler, if not, the value that was given back from the `checkCritTime` method is returned. Everytime data is fetched from the sensor regularly, the cache is being refreshed. Another security feature is the user authentication. The users are compared to the entries of a userdatabase the gateway handler is implementing, and therefore usage of the classes methods is granted or not. Because the authentication is managed by a HTTP construct, everytime a method is called also the user ID is checked. To prevent the time loss that is connected with getting data from the database, users, that were already authenticated during a session, are kept in a local hashtable. Authentication is disabled by setting the `authentication` property in the `gatewayxmlrpc.properties` file to ”0”. If database usage is disabled on the gateway handler, the authentication is automatically disabled. On the client, the `setBasicAuthentication` method of the XML-RPC client must be set for the authentication to function. To entirely turn off the authentication, the `implements AuthenticatedHandler` has to be excluded in the class declaration.

8.1.3. AXIS

Apache's AXIS Webservices communicate via SOAP 1.1(Simple Object Access Protocol). AXIS uses Apache Tomcat as webservice application. Similar as in XML-RPC, the data is wrapped into XML for submittance and therefore interoperational communication is made possible. It uses HTTP as transport protocol. SOAP communication in the Sensation project was developed to guarantee future compatibility with the W3C standards. SOAP messages consist of an envelope which has data of the format of transmitted data, and the body which transports the actual data. Here is an example of how AXIS calls look like.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <ns1:getValueVector xmlns:ns1="http://soapinterop.org/">
      <sensorID xsi:type="xsd:string">Sensor1ESBTemp</testParam>
  </ns1:getValueVector>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The above packet shows the SOAP call to the same method as the XML-RPC call, namely `getValueVector`. It is accompanied by the parameter sensorID, of type String which contains `Sensor1ESBTemp`.
In the webservice creation process a **W**eb **S**ervice **D**efinition **L**anguage (WSDL) file is generated which holds all information needed for creating clients to this specific service by showing the methods that can be run. In the case of Sensation, the WSDL file for one method looks like this:(excerpt)

```
<wsdl:message name="registerAxisRequest">
  <wsdl:part name="ip" type="xsd:string"/>
  <wsdl:part name="sensorID" type="xsd:string"/>
  <wsdl:part name="port" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="registerAxisResponse">
  <wsdl:part name="registerAxisReturn" type="xsd:string"/>
</wsdl:message>
```

The above packet shows the SOAP call to the same method as the XML-RPC call, namely `getValueVector`. It is accompanied by the parameter sensorID, of type String which contains `Sensor1ESBTemp`.
In the webservice creation process a **W**eb **S**ervice **D**efinition **L**anguage (WSDL) file is generated which holds all information needed for creating clients to this specific service by showing the methods that can be run. In the case of Sensation, the WSDL file for one method looks like this:(excerpt)

```
<wsdl:message name="registerAxisRequest">
  <wsdl:part name="ip" type="xsd:string"/>
  <wsdl:part name="sensorID" type="xsd:string"/>
  <wsdl:part name="port" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="registerAxisResponse">
  <wsdl:part name="registerAxisReturn" type="xsd:string"/>
</wsdl:message>
```

In this code snippet is shown, that the `registerAxis` method has an entry for the request as well as for the response. The request needs three parameters: one `String` for the IP, one `String` for a variable called `sensorID` and one `String` for another varibale called `port`. As return value it gets a `String`, what can be seen at the line
`<wsdl:part name="registerAxisReturn" type="xsd:string"/>`
An AXIS webservice is identified by the WSDL by clients, and builds the interface for method requests. Once the webservice is created and deployed on the Tomcat server, its methods can be simply run by creating an instance of the webserver on the client. This process is shown below.

```
private static AxisServer server;

  private static AxisServerService service;

  service = new axisserver.ws.AxisServerServiceLocator();
        try {
          server = service.getAxisServer();
        } catch (Exception ex) {
          System.out.println("Error:  " + e);
        }
```

The `ServiceLocator` is one of the by the `WSDL2Java` generated skeleton classes. It is used for locating the service classes that implement the methods for the service. The server is instantiated via this imported webservice classes. By simply calling the methods that the server implements, those methods can be requested. For more detailed information about how to create webservices and how to use them, consult http://ws.apache.org/axis/java/index.html

8.1.4. Gateway AXIS

The `AxisServer` class builds the main gateway for **S**imple **O**bject **A**ccess **P**rotocol (SOAP) connections. It implements the same basic functions (see 8.1.5) as the XML-RPC gateway. In the `axisserver.properties` file entries can be set to declare the IPs of the AXIS webserver and the gateway handler that is requested for data. Everytime a method request to the AXIS gateway is started, an XML-RPC client to the server on the gateway handler is initialized, and with it, the request is forwarded to the server classes. The communication runs over a SOAP interface only between the client and the AXIS gateway . The AXIS gateway was developed using Apache AXIS 1.1 and Apache Tomcat 5.028.

8.1.5. Functionality of the XML-RPC- and AXIS Gateway

For editing, the gateways can be found under the classname `GatewayXMLRPC` and accordingly `AxisServer`. The `AxisServer` file can be found in the Sensation Project path in the `axisserver.ws` package, the `GatewayXMLRPC` in `src.de.buw.medien.cscw.sensation.server` package. The XML-RPC- and the AXIS gateway both feature nearly the same functionality. The "*" indicates that the request for the method can be done for different datatypes, by replacing it accordingly with `String`, `Vector`, `Hashtable`, `HashMap`, and sometimes `XML`. This was done for increasing the compatibility with client applications. For exact request format information, the specific classes, or the JavaDoc of the project is recommended as reference.

- `authenticate(String user, String pw)` : This method is only implemented in the XML-RPC gateway and is used for authentication. It is called everytime, when another method is requested on the gateway.
- `check()` : This method returns a String containing "running" when the gateway is running.
- `getAllLocations*()` : This method returns all location ids that sensors are located at.
- `getAllSensors*()` : This method returns all sensor IDs no matter of their location.
- `getHardwareMetadata(String hardwareID)` : This method returns the hardware description of a sensor as Vector
- `getSensors*(String locationID)` : This method returns the sensor IDs located at the specific location.

- `getSensorXML(String sensorID)` : This method returns the XML description of the sensor.
- `getServerDescription()` : This method returns the description of the server as String
- `getValue*(String sensorID)` : This method returns the last published value of that specific sensor.
- `getValues*(String sensorID)` : This method is for returning all values of a specific sensor.
- `getValuesXML(String sensorID, String startDate, String endDate)` : This method returns all events of a specific sensor from the starting date to the ending date. It is implemented for the XML-RPC gateway. For date syntax consult the `GatewayXMLRPC` class.
- `getAxisMessage(String ip)` : This method returns the events that happened for registered sensors for AXIS clients. Therefore, its only implemented in the `AxisServer` class.
- `register[Axis](String ip, String sensorID, String port)` : This method is called `registerAxis` for the AXIS gateway, and `register` for the XML-RPC gateway. It's for registering a client for asynchronous services and sensors.
- `unregister[Axis](String ip, String sensorID)`This method is called `unregisterAxis` for the AXIS gateway, and `unregister` for the XML-RPC gateway. It is for unregistering a client for a specific sensor or service.
- `useDB()` This method was only implemented in the XML-RPC gateway and returns a boolean, if the database is running or not.

8.2. PHP

The PHP gateway provides three connection methods for the user:

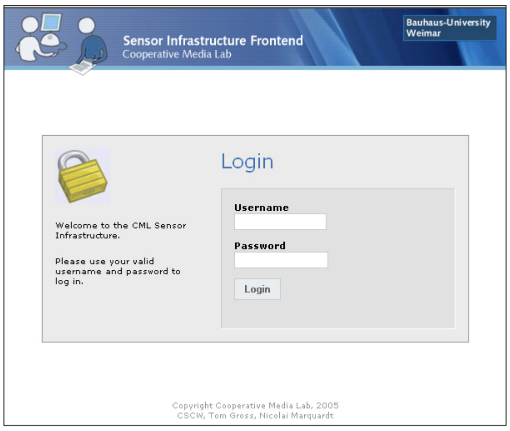
1. **Administration interface:**

You can access the registered locations and sensors of the server, create new locations or sensors and publish sensor values. The additional features are the graph visualization and the export of CSV data.

2. **Mobile interface:**
   A mobile user interface to access the sensor values, display visualizations, create new mobile sensors and publish values.

3. **HTML interface:**
   This is a light-weight HTML interface to the server. All requests and the parameter can send with the HTML address and the server returns only HTML text content.

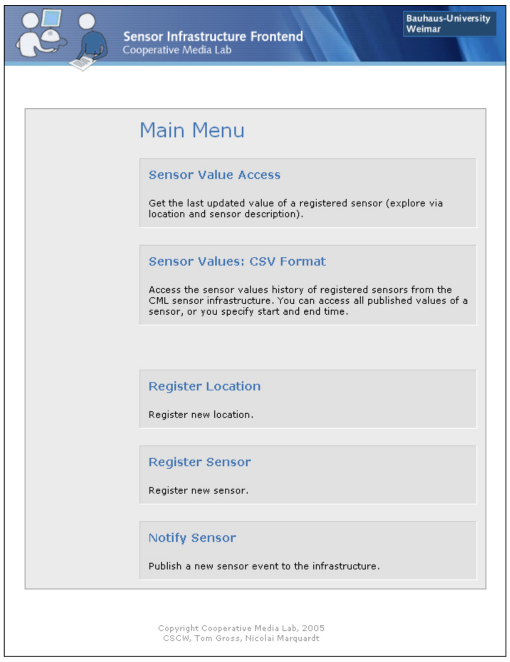
Note: Before using the PHP gateway please check that you have successfully installed the required components (see section 3.4.3) and start the Apache webserver. Then open the login page of the PHP administration interface: `http://[host]/[sensation-directory]/index.php` (figure 8.1).



Welcome to the CML Sensor Infrastructure.

Please use your valid username and password to log in.

Copyright Cooperative Media Lab, 2005
CSCW, Tom Gross, Nicolai Marquardt



Sensor Value Access

Get the last updated value of a registered sensor (explore via location and sensor description).

Sensor Values: CSV Format

Access the sensor values history of registered sensors from the CML sensor infrastructure. You can access all published values of a sensor, or you specify start and end time.

Register Location

Register new location.

Register Sensor

Register new sensor.

Notify Sensor

Publish a new sensor event to the infrastructure.

Copyright Cooperative Media Lab, 2005
CSCW, Tom Gross, Nicolai Marquardt

Figure 8.1.: PHP admin tool: Login window (left) and main menu (right)

Login with the superuser name and password (see chapter 3.4.3 if you want to change the superuser name or password), and you can choose one of the following menu items:

- **Sensor Value Access**
  Display the last updated value of a registered sensor.

- **Sensor Values: CSV Format**
  Select a location and a sensor ID. The PHP method will display all values from start to end time. You can save these values as CSV file and import the values into spreadsheet software (e.g. Microsoft Excel, see chapter 8.2.2).

- **Sensor value visualization (BETA)**
  The PHP script will display the last 30 values of the selected sensor in a graph visualization.

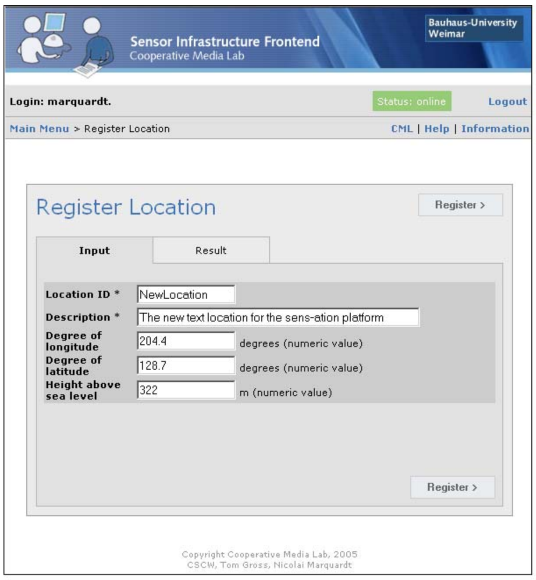
- **Register Location**
  HTML form for the location registration; sends the location description as XML file to the platform.

- **Register Sensor**
  Register a new sensor at the platform: you can specify the sensor type and location as well as the description, owner, native data type and other optional parameter (see figure 8.2).

- **Notify Sensor**
  You can send notification events for each of the registered sensors. This is often used for testing of sensor transmissions.

8.2.1. Server Connection

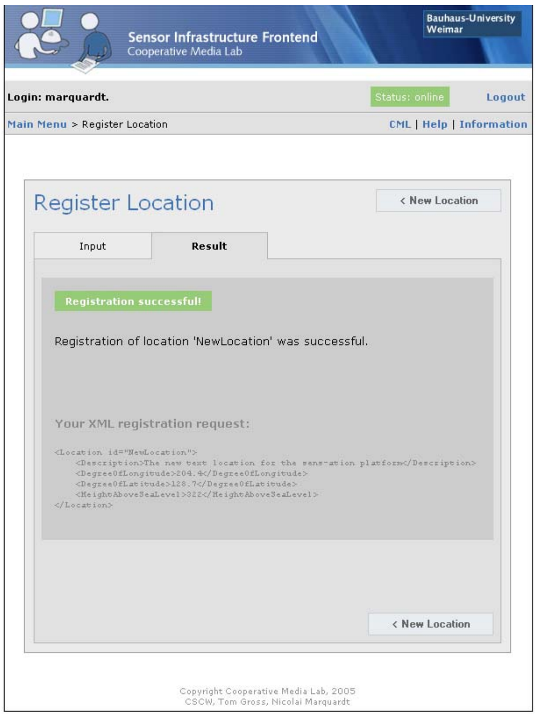
The main methods to establish connections to the server are located in the `functions.inc.php` file. Include this file in the PHP pages which have to transmit data to the server. Furthermore you have to include the `config.inc.php` and `./domit/xml_domit_include.php` file and start each PHP page with the `session_start();` command. The latter is needed to access the session member variables.
In the file `functions.inc.php` you can access the method `getXMLfromServer($method, $param)` to receive XML data (add the remote method name as the first parameter and the parameter vector as the second parameter) from the server (see listing 8.2.1). The script creates a new XML-RPC client instance (line 2) connected to the server and port specified in the session members. The method name is the string of the main service name


Sensor Infrastructure Frontend
Cooperative Media Lab
Bauhaus-University Weimar
Status: online
Logout
Main Menu > Register Location
CML | Help | Information
Register >
Input
Result
Register >
Copyright Cooperative Media Lab, 2005
CSCW, Tom Gross, Nicolai Marquardt

Login: marquardt.

Register Location

Location ID *	NewLocation	
Description *	The new text location for the sens-ation platform	
Degree of longitude	204.4	degrees (numeric value)
Degree of latitude	128.7	degrees (numeric value)
Height above sea level	322	m (numeric value)


Sensor Infrastructure Frontend
Cooperative Media Lab
Bauhaus-University Weimar
Status: online
Logout
Main Menu > Register Location
CML | Help | Information
< New Location
Input
Result
< New Location
Copyright Cooperative Media Lab, 2005
CSCW, Tom Gross, Nicolai Marquardt

Login: marquardt.

Register Location

Registration successfull

Registration of location 'NewLocation' was successful.

Your XML registration request:

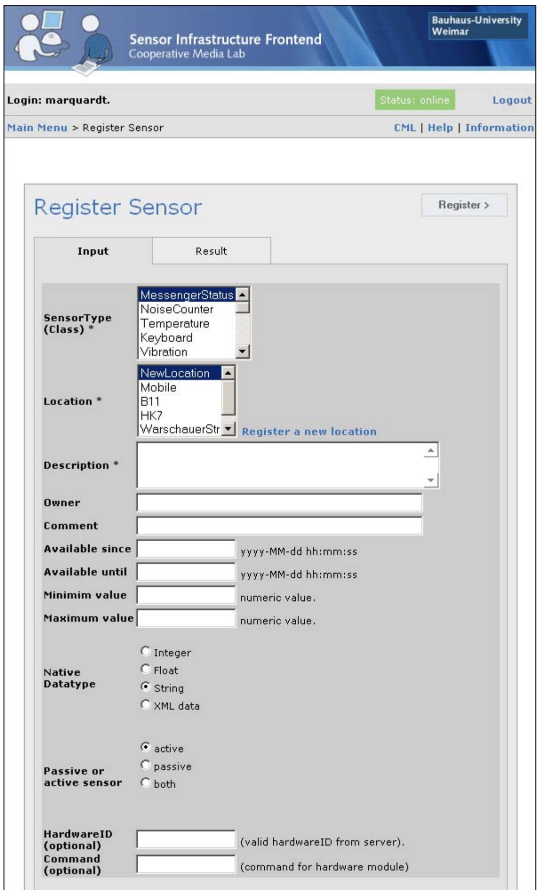
```
<Location id="NewLocation">
    <Description>The new text location for the sens-ation platform</Description>
    <DegreeOfLongitude>204.4</DegreeOfLongitude>
    <DegreeOfLatitude>128.7</DegreeOfLatitude>
    <HeightAboveSeaLevel>322</HeightAboveSeaLevel>
</Location>
```

Figure 8.2.: PHP admin tool: Register location form (left) and result page (right)

(`GatewayXMLRPC`) and the method parameter (the complete string is for example `GatewayXMLRPC.getSensorsXML`). Then a new call with the parameters is created (line 5) and the script opend the connection to the server (line 6). At the end of the method, the response string is converted to a XML document (line 10).

```
1  function getXMLfromServer($method, $param){
2    $xmlrpc_client = new xmlrpc\_client("/", $_SESSION['rpc_server'], $_SESSION['
       rpc_port']);
3    $method = $_SESSION['rpc_service'] . "." . $method;
4
5    $call = new xmlrpcmsg($method, $param);
6    $response = $xmlrpc_client->send($call);
7
8    $value = $response->value();
9    $convert_string = trim($value->scalarval());
10   return convertToXMLDocument($convert_string);
11 }
```

The `convertToXMLDocument()` replaces the characters for XML braces, creates a new XML document and parses the XML string.


Sensor Infrastructure Frontend
Cooperative Media Lab
Bauhaus-University Weimar

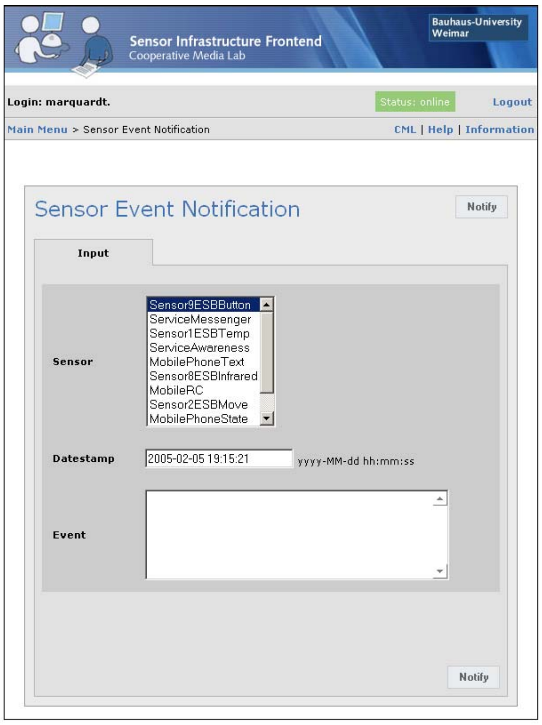
Login: marquardt. Status: online Logout

Main Menu > Register Sensor CML | Help | Information

Register Sensor Register >

Input Result

SensorType (Class) *	MessengerStatus, NoiseCounter, Temperature, Keyboard, Vibration
Location *	NewLocation, Mobile, B11, HK7, WarschauerStr — Register a new location
Description *	
Owner	
Comment	
Available since	yyyy-MM-dd hh:mm:ss
Available until	yyyy-MM-dd hh:mm:ss
Minimim value	numeric value.
Maximum value	numeric value.
Native Datatype	Integer, Float, [✓ String], XML data
Passive or active sensor	[✓ active], passive, both
HardwareID (optional)	(valid hardwareID from server).
Command (optional)	(command for hardware module)


Sensor Infrastructure Frontend
Cooperative Media Lab
Bauhaus-University Weimar

Login: marquardt. Status: online Logout

Main Menu > Sensor Event Notification CML | Help | Information

Sensor Event Notification Notify

Input

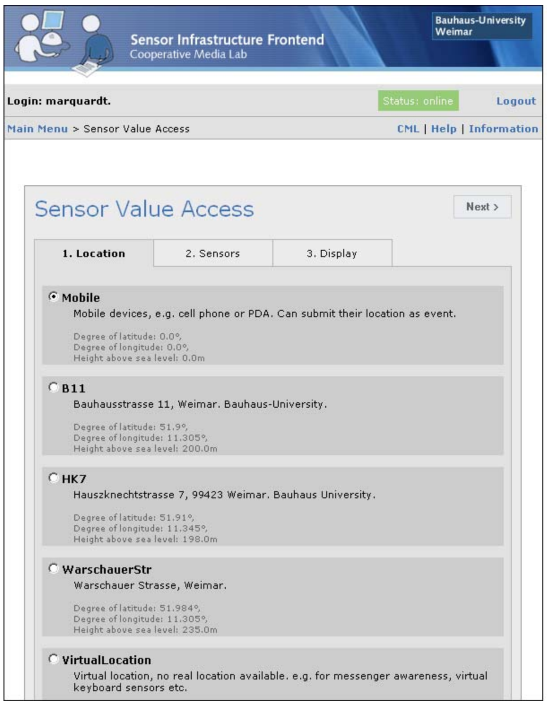
Sensor	Sensor9ESBButton, ServiceMessenger, Sensor1ESBTemp, ServiceAwareness, MobilePhoneText, Sensor8ESBInfrared, MobileRC, Sensor2ESBMove, MobilePhoneState
Datestamp	2005-02-05 19:15:21 yyyy-MM-dd hh:mm:ss
Event	

Notify

Figure 8.3.: PHP admin tool: Register sensor form (left) and event notification form (right)

```
function convertToXMLDocument($toConvert){
  $toConvert = str_replace("&lt;","<",$toConvert);
  $toConvert = str_replace("&gt;",">",$toConvert);
  $document =& new DOMIT_Document();
  $success = $document->parseXML($toConvert, true);
  if($success == 1){
    return $locations;
  }
  return null;
}
```

The other communication methods in the `functions.inc.php` file are working in a similar way. These are the methods `registerOnServer($method, $param)` for the sensor and location registration process and `getDataFromServer($method, $param)` for retrieving data from the server.



Sensor Infrastructure Frontend
Cooperative Media Lab
Bauhaus-University Weimar

Login: marquardt. Status: online Logout

Main Menu > Sensor Value Access CML | Help | Information

Sensor Value Access

Next >

1. Location | 2. Sensors | 3. Display

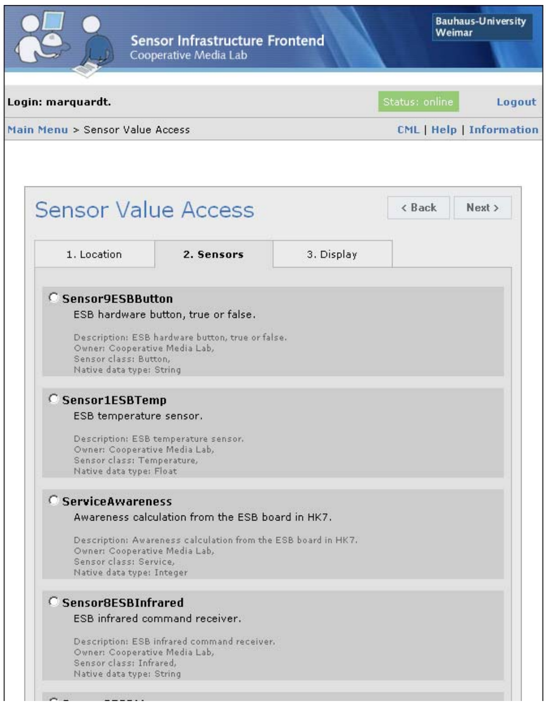
(•) **Mobile**
Mobile devices, e.g. cell phone or PDA. Can submit their location as event.
Degree of latitude: 0.0°,
Degree of longitude: 0.0°,
Height above sea level: 0.0m

( ) **B11**
Bauhausstrasse 11, Weimar. Bauhaus-University.
Degree of latitude: 51.9°,
Degree of longitude: 11.305°,
Height above sea level: 200.0m

( ) **HK7**
Hauszknechtstrasse 7, 99423 Weimar. Bauhaus University.
Degree of latitude: 51.91°,
Degree of longitude: 11.345°,
Height above sea level: 198.0m

( ) **WarschauerStr**
Warschauer Strasse, Weimar.
Degree of latitude: 51.984°,
Degree of longitude: 11.305°,
Height above sea level: 235.0m

( ) **VirtualLocation**
Virtual location, no real location available. e.g. for messenger awareness, virtual keyboard sensors etc.



Sensor Infrastructure Frontend
Cooperative Media Lab
Bauhaus-University Weimar

Login: marquardt. Status: online Logout

Main Menu > Sensor Value Access CML | Help | Information

Sensor Value Access

< Back Next >

1. Location | 2. Sensors | 3. Display

( ) **Sensor9ESBButton**
ESB hardware button, true or false.
Description: ESB hardware button, true or false.
Owner: Cooperative Media Lab,
Sensor class: Button,
Native data type: String

( ) **Sensor1ESBTemp**
ESB temperature sensor.
Description: ESB temperature sensor.
Owner: Cooperative Media Lab,
Sensor class: Temperature,
Native data type: Float

( ) **ServiceAwareness**
Awareness calculation from the ESB board in HK7.
Description: Awareness calculation from the ESB board in HK7.
Owner: Cooperative Media Lab,
Sensor class: Service,
Native data type: Integer

( ) **Sensor8ESBInfrared**
ESB infrared command receiver.
Description: ESB infrared command receiver.
Owner: Cooperative Media Lab,
Sensor class: Infrared,
Native data type: String

Figure 8.4.: PHP admin tool: Sensor value access, select location (left) and the desired sensor from the list (right)

8.2.2. CSV Data and Excel Import

You can choose the menu point "Sensor Values: CSV Format" to get a list of sensor values in CSV[1] format. Select the location on the first tab page and the desired sensor on the second tab. The values will be read from the server database and displayed in the textbox of the third HTML tab page website. You can change the period of time of the sensor values you need.
To import the CSV values into a spreadsheet software like Microsoft Excel, just follow the subsequent procedure:

1. Copy the values in the textbox (Windows: Ctrl+A, Ctrl+C. OS X: Apple+A, Apple+C), open a new textfile in the editor (e.g. TextEdit, Notepad), paste the data (Windows: Ctrl-V. OS X: Apple+V) and save the textfile as `data.csv`.

2. Open Microsoft Excel and open a new spreadsheet document.

[1]CSV = Comma Separated Value File Format

3. In the `Data` menu select `Import data...`

4. In the file dialog choose directory you have saved the CSV file and select the file: `data.csv`. Click `OK`.

5. Click `Next` and select on the second import assistant page the "Comma" as separation character. Finish the import assistant dialog by clicking `Finish`.

6. The data should now be imported into the excel spreadsheet file.

7. Select the first column (by clicking the "A" in the headline), right-click the column headline and select the menu point `Cell Format...`.

8. In the textfield for the data format insert: `JJ-MM-TT hh:mm:ss` (figure 8.5) and click the `OK` button.

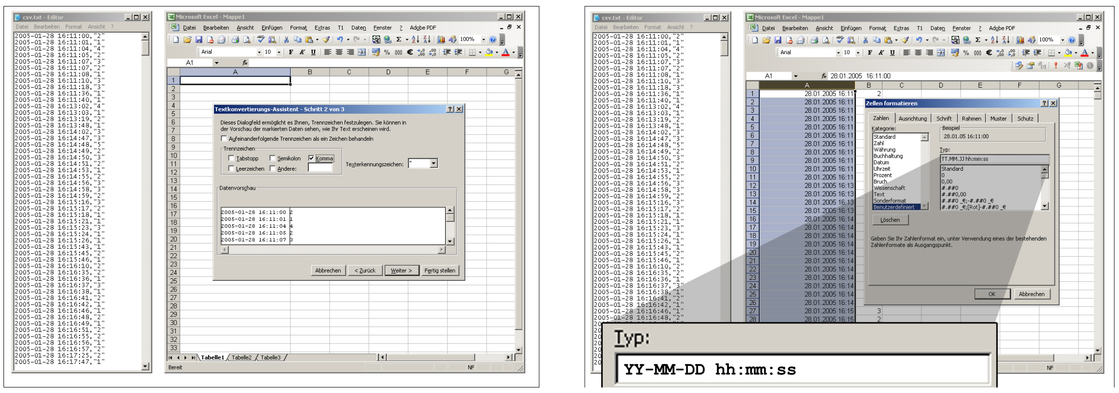


Figure 8.5.: Excel CSV import: CSV import assistant dialog (left) and the data format dialog (right)

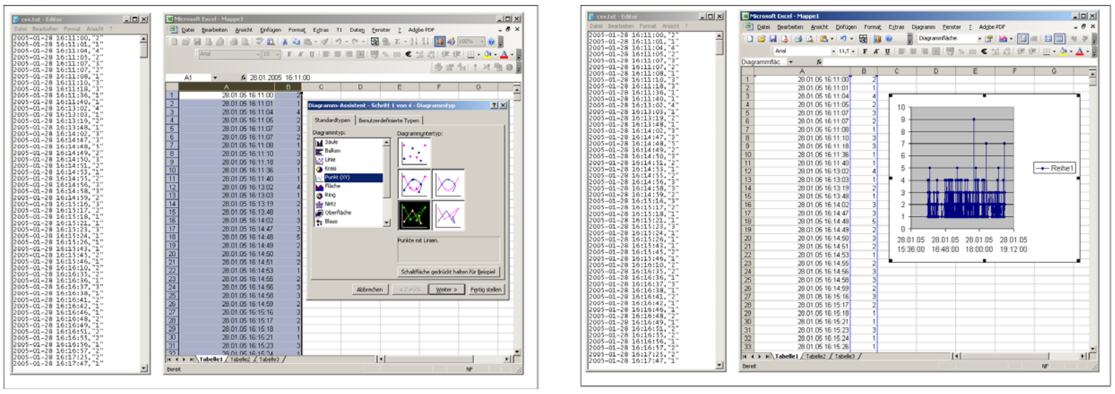
The CSV data is now successfully imported into your Excel spreadsheet document. You can now generate a graph viualization (figure 8.6):

1. Select the two columns you have imported before (clicking the headline of the first column and drag the mouse pointer to the second column).

2. Select the graph icon in the toolbar

3. In the graph type dialog window select the "Points (XY)" graph on the left side and the "Connected lines" viualization on the right side.

4. Finish the graph assistant dialog.



```
2005-01-28 16:11:00,"2"
2005-01-28 16:11:01,"1"
2005-01-28 16:11:04,"4"
2005-01-28 16:11:05,"2"
2005-01-28 16:11:07,"3"
2005-01-28 16:11:07,"2"
2005-01-28 16:11:08,"1"
2005-01-28 16:11:10,"3"
2005-01-28 16:11:18,"3"
2005-01-28 16:11:36,"1"
2005-01-28 16:11:40,"1"
2005-01-28 16:13:02,"4"
2005-01-28 16:13:03,"1"
2005-01-28 16:13:19,"2"
2005-01-28 16:13:48,"1"
2005-01-28 16:14:02,"3"
2005-01-28 16:14:47,"3"
2005-01-28 16:14:48,"5"
2005-01-28 16:14:49,"2"
2005-01-28 16:14:50,"3"
2005-01-28 16:14:51,"2"
2005-01-28 16:14:53,"1"
2005-01-28 16:14:55,"2"
2005-01-28 16:14:56,"3"
2005-01-28 16:14:58,"3"
2005-01-28 16:14:59,"2"
2005-01-28 16:15:16,"3"
2005-01-28 16:15:17,"2"
2005-01-28 16:15:18,"1"
2005-01-28 16:15:21,"1"
2005-01-28 16:15:23,"3"
2005-01-28 16:15:24,"1"
2005-01-28 16:15:26,"1"
2005-01-28 16:15:43,"1"
2005-01-28 16:15:45,"2"
2005-01-28 16:15:46,"1"
2005-01-28 16:16:10,"2"
2005-01-28 16:16:35,"2"
2005-01-28 16:16:36,"1"
2005-01-28 16:16:37,"3"
2005-01-28 16:16:38,"1"
2005-01-28 16:16:41,"2"
2005-01-28 16:16:42,"1"
2005-01-28 16:16:46,"1"
2005-01-28 16:16:48,"2"
2005-01-28 16:16:49,"1"
2005-01-28 16:16:51,"2"
2005-01-28 16:16:55,"2"
2005-01-28 16:16:56,"1"
2005-01-28 16:16:57,"2"
2005-01-28 16:17:25,"2"
2005-01-28 16:17:47,"1"
```

A1 28.01.2005 16:11:00

	A	B
1	28.01.05 16:11:00	2
2	28.01.05 16:11:01	1
3	28.01.05 16:11:04	4
4	28.01.05 16:11:05	2
5	28.01.05 16:11:07	3
6	28.01.05 16:11:07	2
7	28.01.05 16:11:08	1
8	28.01.05 16:11:10	3
9	28.01.05 16:11:18	3
10	28.01.05 16:11:36	1
11	28.01.05 16:11:40	1
12	28.01.05 16:13:02	4
13	28.01.05 16:13:03	1
14	28.01.05 16:13:19	2
15	28.01.05 16:13:48	1
16	28.01.05 16:14:02	3
17	28.01.05 16:14:47	3
18	28.01.05 16:14:48	5
19	28.01.05 16:14:49	2
20	28.01.05 16:14:50	3
21	28.01.05 16:14:51	2
22	28.01.05 16:14:53	1
23	28.01.05 16:14:55	2
24	28.01.05 16:14:56	3
25	28.01.05 16:14:58	3
26	28.01.05 16:14:59	2
27	28.01.05 16:15:16	3
28	28.01.05 16:15:17	2
29	28.01.05 16:15:18	1
30	28.01.05 16:15:21	1
31	28.01.05 16:15:23	3
32	28.01.05 16:15:24	1

Figure 8.6.: Excel graph visualization: The graph type dialog (left) and the finished graph (right)

In figure 8.7 and 8.8 are some examples of the generated graph visualizations.

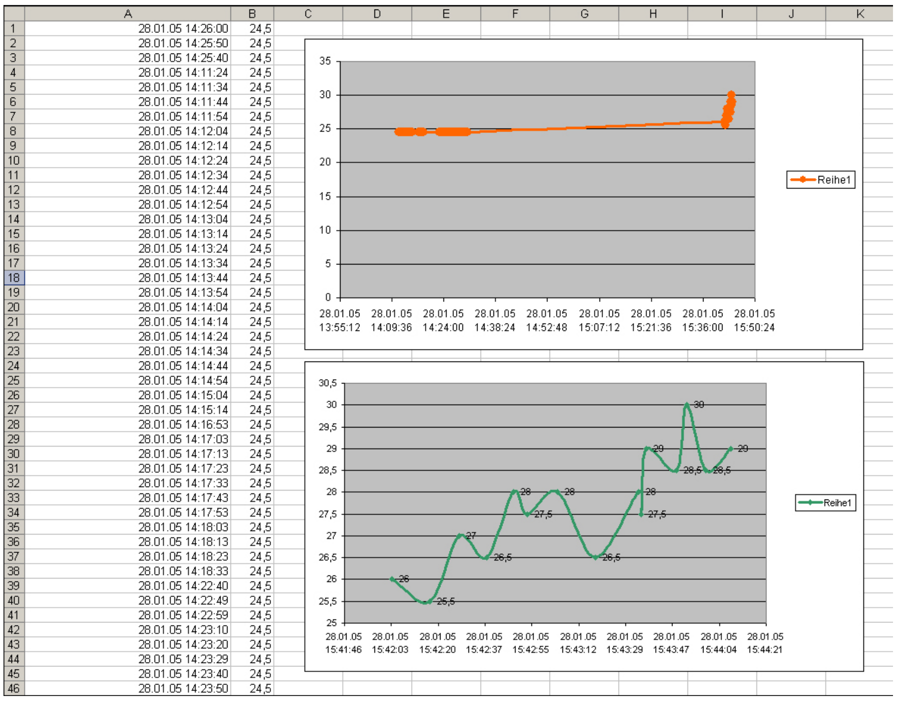
8.2.3. Mobile Portal

As a different method for mobile device to connect to the server infrastructure we developed a mobile portal. To start the portal website open the file `[ApacheIP]:[ApachePort]/mobile.php`. The portal is created with the PHP scripting language and HTML code is created[2].
The figures 8.9 and 8.10 illustrate the screens of the mobile device portal page. The methods for data access are the same as in the main PHP classes.

8.3. HTML

The HTML gateway is a light-weight access method to the platform. This gateway enables the access to location information, sensor discovery and sensor values via the GET method of a HTTP request. For clients that can not send XML-RPC requests or with low capacity CPUs (and therefore slow XML parsers)

[2]Instead of creating HTML content (that is only accessible of mobile devices with an state of the art webbrowser) it is conceivable to create WAP pages. This could be implemented in future releases of the Sens-ation platform.



	A	B
1	28.01.05 14:26:00	24,5
2	28.01.05 14:25:50	24,5
3	28.01.05 14:25:40	24,5
4	28.01.05 14:11:24	24,5
5	28.01.05 14:11:34	24,5
6	28.01.05 14:11:44	24,5
7	28.01.05 14:11:54	24,5
8	28.01.05 14:12:04	24,5
9	28.01.05 14:12:14	24,5
10	28.01.05 14:12:24	24,5
11	28.01.05 14:12:34	24,5
12	28.01.05 14:12:44	24,5
13	28.01.05 14:12:54	24,5
14	28.01.05 14:13:04	24,5
15	28.01.05 14:13:14	24,5
16	28.01.05 14:13:24	24,5
17	28.01.05 14:13:34	24,5
18	28.01.05 14:13:44	24,5
19	28.01.05 14:13:54	24,5
20	28.01.05 14:14:04	24,5
21	28.01.05 14:14:14	24,5
22	28.01.05 14:14:24	24,5
23	28.01.05 14:14:34	24,5
24	28.01.05 14:14:44	24,5
25	28.01.05 14:14:54	24,5
26	28.01.05 14:15:04	24,5
27	28.01.05 14:15:14	24,5
28	28.01.05 14:16:53	24,5
29	28.01.05 14:17:03	24,5
30	28.01.05 14:17:13	24,5
31	28.01.05 14:17:23	24,5
32	28.01.05 14:17:33	24,5
33	28.01.05 14:17:43	24,5
34	28.01.05 14:17:53	24,5
35	28.01.05 14:18:03	24,5
36	28.01.05 14:18:13	24,5
37	28.01.05 14:18:23	24,5
38	28.01.05 14:18:33	24,5
39	28.01.05 14:22:40	24,5
40	28.01.05 14:22:49	24,5
41	28.01.05 14:22:59	24,5
42	28.01.05 14:23:10	24,5
43	28.01.05 14:23:20	24,5
44	28.01.05 14:23:29	24,5
45	28.01.05 14:23:40	24,5
46	28.01.05 14:23:50	24,5

Figure 8.7.: Excel graph visualization: Temperature variation (overview on the top, detail view on the bottom)

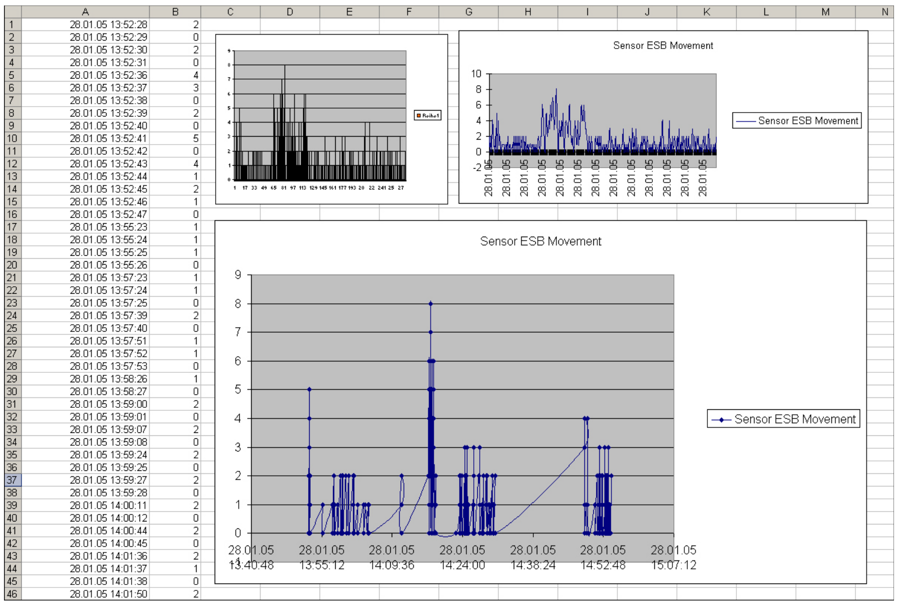
this is a access method to the infrastructure without the overhead of XML data. Access the HTML gateway with the address `http://[host]/[sensation-directory]/ port.php`, the commands are specified with the `commands` parameter and all additional parameters are added to the address.

- `port.php?command=getvalue&sensorid=[id]`
  Get the last submitted value of the selected sensor. Only the event value will be displayed (without datestamp or sensorID)

- `port.php?command=getsensors`
  Get all registered sensors, as string separated list (e.g. "*Sensor*1|*Sensor*2|*AnotherSensor*").

- `port.php?command=getlocations`
  Get all registered locations, as string separated list (e.g. "Location1—Location2").

- `port.php?command=notify&sensorid=[id]&event=[message]`
  Notification method for new sensor events; the missing datestamp will be



	A	B
1	28.01.05 13:52:28	2
2	28.01.05 13:52:29	0
3	28.01.05 13:52:30	2
4	28.01.05 13:52:31	0
5	28.01.05 13:52:36	4
6	28.01.05 13:52:37	3
7	28.01.05 13:52:38	0
8	28.01.05 13:52:39	2
9	28.01.05 13:52:40	0
10	28.01.05 13:52:41	5
11	28.01.05 13:52:42	0
12	28.01.05 13:52:43	4
13	28.01.05 13:52:44	1
14	28.01.05 13:52:45	2
15	28.01.05 13:52:46	1
16	28.01.05 13:52:47	0
17	28.01.05 13:55:23	1
18	28.01.05 13:55:24	1
19	28.01.05 13:55:25	1
20	28.01.05 13:55:26	0
21	28.01.05 13:57:23	1
22	28.01.05 13:57:24	1
23	28.01.05 13:57:25	0
24	28.01.05 13:57:39	2
25	28.01.05 13:57:40	0
26	28.01.05 13:57:51	1
27	28.01.05 13:57:52	1
28	28.01.05 13:57:53	0
29	28.01.05 13:58:26	1
30	28.01.05 13:58:27	0
31	28.01.05 13:59:00	2
32	28.01.05 13:59:01	0
33	28.01.05 13:59:07	2
34	28.01.05 13:59:08	0
35	28.01.05 13:59:24	2
36	28.01.05 13:59:25	0
37	28.01.05 13:59:27	2
38	28.01.05 13:59:28	0
39	28.01.05 14:00:11	2
40	28.01.05 14:00:12	0
41	28.01.05 14:00:44	2
42	28.01.05 14:00:45	0
43	28.01.05 14:01:36	2
44	28.01.05 14:01:37	1
45	28.01.05 14:01:38	0
46	28.01.05 14:01:50	2

Figure 8.8.: Excel graph visualization: Movement detection sensor, three graph variations

completed with the current server time.

- `port.php?command=notify& sensorid=[id]& datestamp=[iso8601-date]]& event=[message]`
  Notification method for new sensor events, with the datestamp in ISO-8601 format (yyyy-MM-dd hh:mm:ss, e.g. 2005-12-30 22:12:17) [ISO 8601 Date and Time]

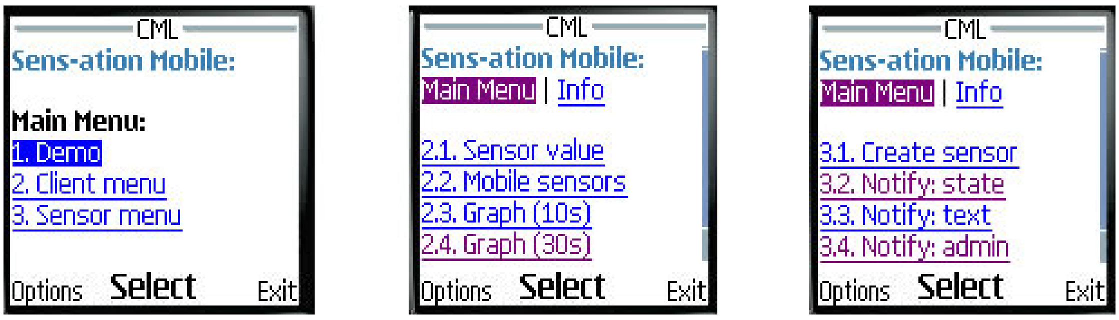
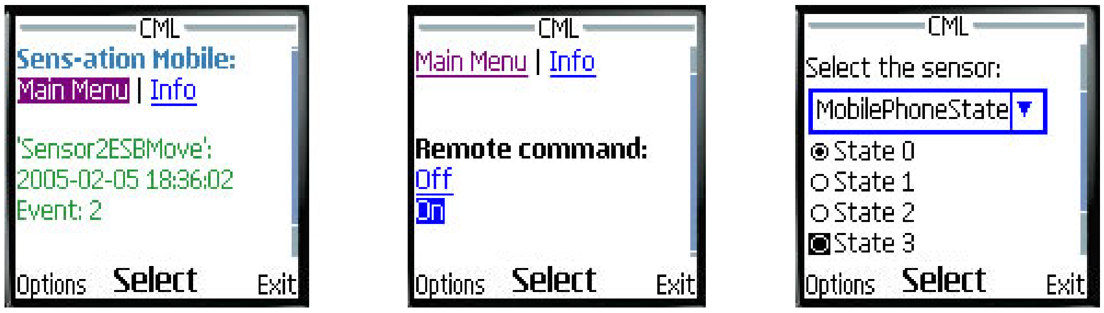


Figure 8.9.: Mobile portal: start page (left), client menu (centre) and sensor menu (right)



Sens-ation Mobile:

'Sensor2ESBMove':
2005-02-05 18:36:02
Event: 2

Remote command:
Off
On

Select the sensor:
MobilePhoneState ▾
● State 0
○ State 1
○ State 2
■ State 3

Figure 8.10.: Mobile portal: sensor value access (left), remote control (centre) and sensor notification form (right)

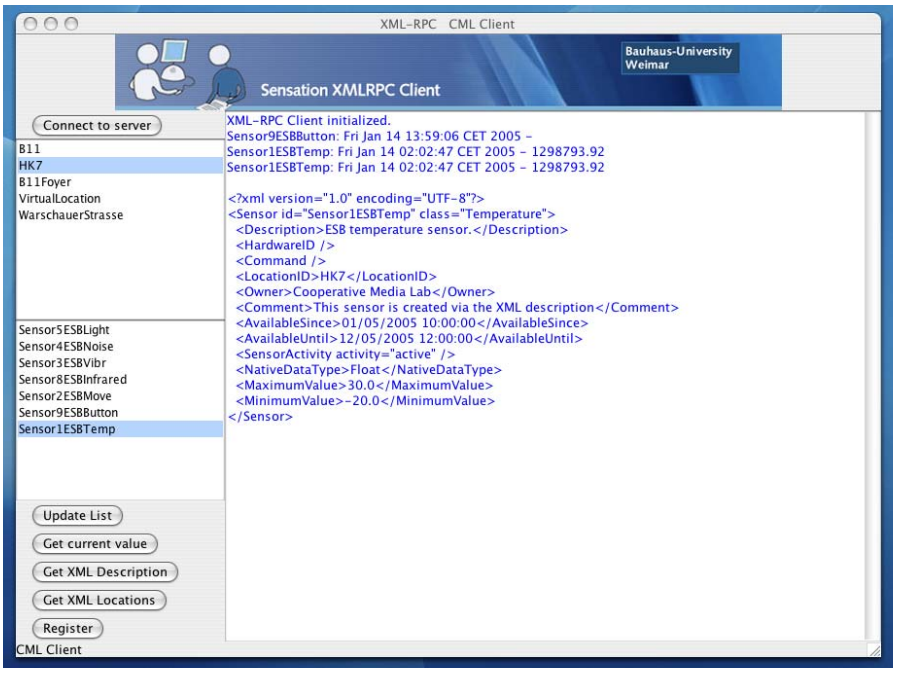
This gateway can easily be extended with additional methods, e.g. more complex sensor discovery or sensor/location registry. Furthermore an integration of security functions will be an essential part of future extensions.

9. Clients

We have developed various clients for Mac OS X and Windows platform to demonstrate the easy access of clients to sensor values at the platform. We have implemented these clients with Java, AppleScript OS X scripting language and J2ME.

9.1. XML-RPC

The XML-RPC client (`XMLRPCClientGUI`) is a rapid prototyping development. It consists of a main GUI which has on the left side the interaction interface and on the right side the text window for user feedback. As the name says, it communicates with the server via the XML-RPC gateway. In the top textfield the adress and the port of the XML-RPC gateway have to be entered. If you click "Connect" and no error occures, the other buttons are enabled for interaction. On the other buttons basic interaction with the server is made possible. You can get a list of sensors and get values that the sensors have published. A special feature is resembled in the "register" button: The client is not only initialized as a client to the XML-RPC gateway, but also as Webserver to be able to receive calls. The resulting observer pattern is used in the notification mechanism, that tells clients of events that happened on sensors that they were registering for. In the `register` method, the IP and the port of the registrating client are sent with the name of the sensor to the XML-RPC gateway. The gateway handler which finally receives the call, is able to build up a backing call with the sent IP and port to the client. This is done by initializing an XML-RPC client on the XML-RPC gateway, which contacts the waiting XML-RPC server on the client. A security mechanism prevents the user from quitting the client without unregistering. Everytime a user is registering for a sensor or service, its name is put in a local Hashtable. As soon as the program is quitted, the "exit" method takes effect by checking first the local registry, if there is any sensor still registered, and unregisters for them automatically. The XML-RPC client needs as libraries: XML-RPC 1.1, and the `SensorValue` class from the `Server.jar` package.



```
XML-RPC Client initialized.
Sensor9ESBButton: Fri Jan 14 13:59:06 CET 2005 -
Sensor1ESBTemp: Fri Jan 14 02:02:47 CET 2005 - 1298793.92
Sensor1ESBTemp: Fri Jan 14 02:02:47 CET 2005 - 1298793.92

<?xml version="1.0" encoding="UTF-8"?>
<Sensor id="Sensor1ESBTemp" class="Temperature">
 <Description>ESB temperature sensor.</Description>
 <HardwareID />
 <Command />
 <LocationID>HK7</LocationID>
 <Owner>Cooperative Media Lab</Owner>
 <Comment>This sensor is created via the XML description</Comment>
 <AvailableSince>01/05/2005 10:00:00</AvailableSince>
 <AvailableUntil>12/05/2005 12:00:00</AvailableUntil>
 <SensorActivity activity="active" />
 <NativeDataType>Float</NativeDataType>
 <MaximumValue>30.0</MaximumValue>
 <MinimumValue>-20.0</MinimumValue>
</Sensor>
```

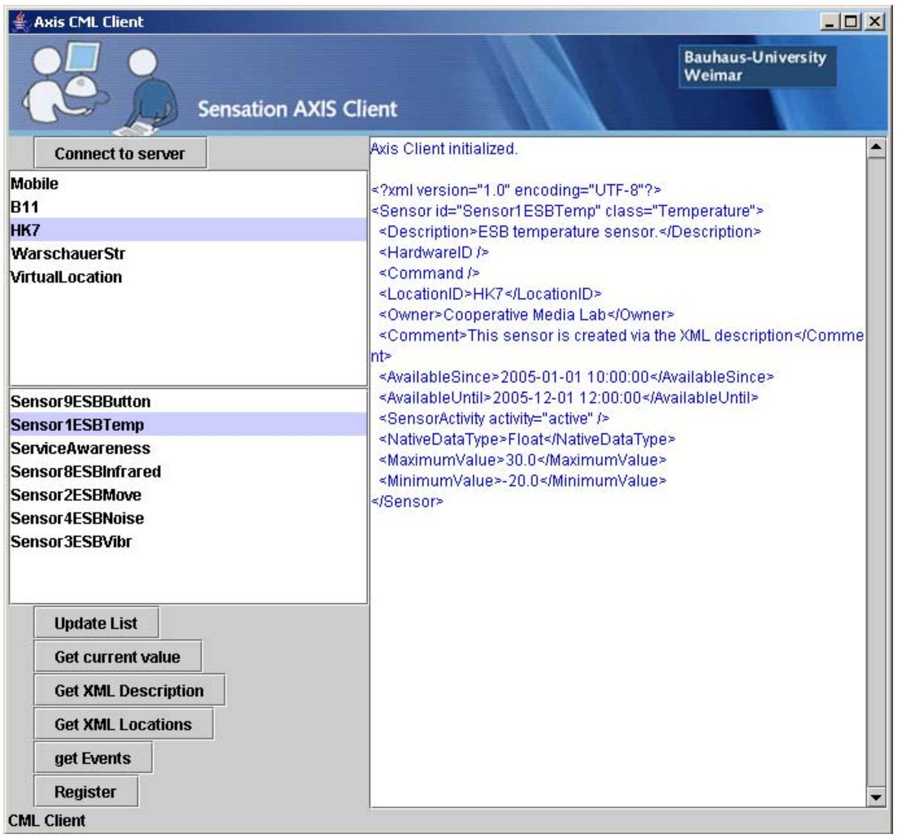
Figure 9.1.: The XML-RPC client GUI

9.2. AXIS

The AXIS client is made of a nearly similiar GUI and also features nearly the same functionality as the XML-RPC client. Because it needs to know where the gateway is, it has a `axisserver.properties` file attached, that tells the server class, where to find the AXIS gateway and the gateway handler. The notification is passive, that means that you will have actively to click the "get Events" button to get the events that were saved on the server in the notifaction process. The AXIS client needs the following libraries: XML-RPC 1.2b1, AXIS 1.1, Xerces 2.5.0, XStream 1.0.2, JDOM 1.0, and the with the packet included `axisserver.jar` which contains the AXIS gateway classes.

9.3. Moving Awareness

These are also Java clients, which using the XML-RPC gateway to get sensor values and exec calls for execute internal AppleScripts. 'Observer' in the name of the clients refer to the operating range and not to the observer pattern. They



```
Axis Client initialized.

<?xml version="1.0" encoding="UTF-8"?>
<Sensor id="Sensor1ESBTemp" class="Temperature">
  <Description>ESB temperature sensor.</Description>
  <HardwareID />
  <Command />
  <LocationID>HK7</LocationID>
  <Owner>Cooperative Media Lab</Owner>
  <Comment>This sensor is created via the XML description</Comme
nt>
  <AvailableSince>2005-01-01 10:00:00</AvailableSince>
  <AvailableUntil>2005-12-01 12:00:00</AvailableUntil>
  <SensorActivity activity="active" />
  <NativeDataType>Float</NativeDataType>
  <MaximumValue>30.0</MaximumValue>
  <MinimumValue>-20.0</MinimumValue>
</Sensor>
```

Figure 9.2.: The AXIS client GUI

monitors the movement in a remote location by controlling the speed of an animation or, if they runns on a Mac OX, the volume of the computer.

The clients looks similar, but there's a decisive difference. The `PollingClient` requests the sensor values from the server in a specified timeinterval. Therefore responsible is the `run` method. This method requests the present values, set the variable, which control the speed or the volume and calls the methods for the feedback (`getSoundEvent`, `getVisuEvent`).
The `ListeningClient` registers 4.4 themself for the events of a sensor and listens to the set port for the sensor values. So it has got the methods `register` and `unregister` and for the listening this client instantiate a `WebServer`. The method `notify` gets the sensor values and calls the feedback methods.

9.4. Chart

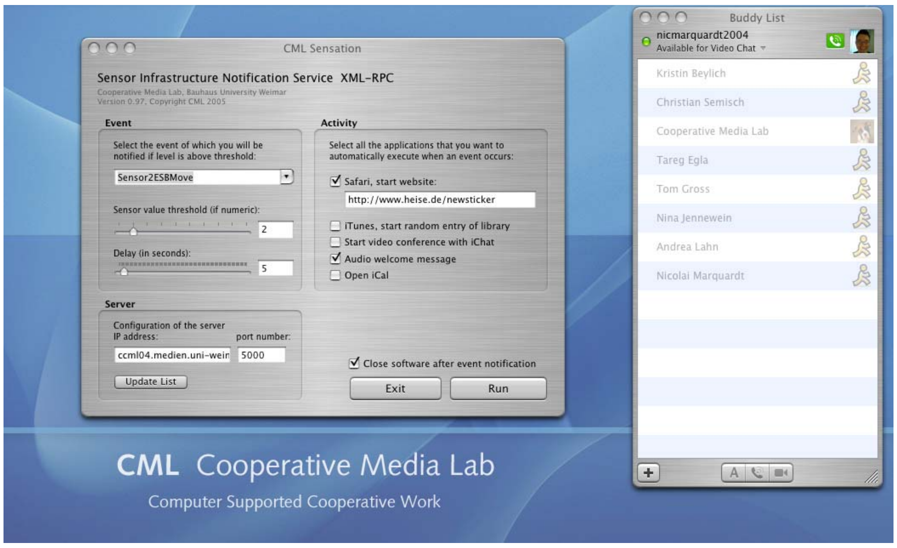
The chart client uses XML-RPC to register to all available sensors and is notified everytime an event happens. It's functionality is adapted from the XML-RPC client. It lists the amount of events that give back integers in comparison to their occurrence in percent in a pie chart. It also displays the level of noise and movement in three steps: high, intermediate, and low, to give an impression of what kind of atmosphere has been resolving in the room. The same is done with the movement values in the room, also represented in three bars, high, intermediate movement and low movement.

9.5. AppleScript

Beyond the Java software clients we have implemented a client application written in the Apple OS X scripting language: AppleScript [AppleScript Documentation]. With AppleScript you can use the scripting port of many standard OS X applications, e.g. to automate iTunes (play music), Safari (open website) or iChat (start video conference). To illustrate the connection between these operating system scripting functions and our platform we have been developed the following software tool.

9.5.1. Notification Service

The notification service tool is a connector between the Sens-ation platform and the OS X operating system. It uses the scripting commands to control the system applications of OS X, e.g. Safari webbrowser, iChat or iCal.
On the left side of the tool dialog you can choose the sensor that 'activates' the system event. This sensor will be observed and if an event occurs, the notififcation service tool activates (or controls) the selected application. The right side of the tool dialog is to specify the scripted application: you can start Safari browser window and open a website, play a audio message or start the iCal calendar software. To start the application click the `Run` button (but please verify that you have entered the correct server in the lower left corner of the dialog window).


CML Sensation

Sensor Infrastructure Notification Service XML-RPC

Cooperative Media Lab, Bauhaus University Weimar
Version 0.97, Copyright CML 2005

Event

Select the event of which you will be notified if level is above threshold:
Sensor2ESBMove

Sensor value threshold (if numeric): 2

Delay (in seconds): 5

Activity

Select all the applications that you want to automatically execute when an event occurs:

- [✓] Safari, start website: http://www.heise.de/newsticker
- [ ] iTunes, start random entry of library
- [ ] Start video conference with iChat
- [✓] Audio welcome message
- [ ] Open iCal

Server

Configuration of the server
IP address: ccml04.medien.uni-weir   port number: 5000

- [✓] Close software after event notification

Buddy List

nicmarquardt2004
Available for Video Chat

- Kristin Beylich
- Christian Semisch
- Cooperative Media Lab
- Tareg Egla
- Tom Gross
- Nina Jennewein
- Andrea Lahn
- Nicolai Marquardt

Update List
Exit
Run
CML Cooperative Media Lab
Computer Supported Cooperative Work

Figure 9.3.: AppleScript notification service

9.5.2. XML-RPC Connection

For the connection to the Sens-ation server from within the AppleScript application, we use the `call xmlrpc` method. In listing 9.1 you can see the `getValue` method of the notification service script. With the statement `tell application [server:port]` (line 5) you can establish the connection to the server and execute the XML-RPC call (line 6). Specify the two needed parameters: the remote class module (`GatewayXMLRPC`) and the method name (`getValueString`). The result of the XML-RPC call will be stored in the `this_result` variable and we return this result with a true statement (line 9). If any error occurs, the `on error` block (line 10) will be executed: the method returns the error message and the false statement. The developers manual for XML-RPC and SOAP programming with AppleScript can be found at [AppleScript XML-RPC and SOAP].

Listing 9.1: Initiate a XML-RPC connection using AppleScript

```
1  on getValue(sensorID)
2    try
3      set method_parameters to sensorID
4      using terms from application "http://www.apple.com/placebo"
5        tell application "http://" & IP & ":" & Port
6          set this_result to call xmlrpc {method name:"GatewayXMLRPC.
   getValueString", parameters:method_parameters}
7        end tell
8      end using terms from
9      return {true, this_result}
10   on error error_message number error_number
11     set the error_message to "The script was unable to establish a connection"
12     return {false, error_message}
```

```
13     end try
14   end getValue
```

In a similar way the script will connect to the Sens-ation server to retrieve the locations and available sensors.

9.5.3. Scripting Applications

The second part of the script, beyond the retrieving of sensor values, is the scripting of the Mac OS X applications dependent on the values. The scripting process (listing 9.2): Start with the `tell application "[application]"` statement (line 2, here the iChat application[1]) and end with the `end tell` statement (line 8). Between add the application scripting commands, e.g. `activate` for starting the application, or `send video invitation to account "[name]"`. More information about the scripting: [AppleScript Documentation].

Listing 9.2: Scripting the iChat application

```
1  on startIChat()
2    tell application "iChat"
3      activate
4      log in
5      set status message to "Available for Video Chat"
6      set status to available
7      send video invitation to account "Cooperative Media Lab"
8    end tell
9  end startIChat
```

With the second example the safari webbrowser is scripted (listing 9.3). The parameter of the function (line 1) is the string of the URL that has to be opened. Then the function validates that the URL is not already in a open window (line 1-8). If the script can not find the URL, a new browser window will be opened (line 12-13).

Listing 9.3: Scripting safari webbrowser

```
1  on openWebsite(urlName)
2    tell application "Safari"
3      set isAlreadyOpen to false
4      set urlList to URL of every document
5      repeat with urlElement in urlList
6        if urlElement is equal to the urlName then
7          set isAlreadyOpen to true
8        end if
```

[1]An overview of the scriptable apllications of OS X can be found at: `http://www.apple.com/applescript/apps/`

```
    end repeat

    if isAlreadyOpen is false then
      make new document
      set the URL of document 1 to urlName
    end if
  end tell
end openWebsite
```

9.6. Clients: Mobile Client

9.6.1. Introduction

In this documentation I will describe first, what is to be considered to start the program and which technologies were used. In the second part I will explain the implementation of this program.

9.6.2. Installation

Recommend components.

Eclipse 3.0 or Eclipse 3.1 [Eclipse Foundation]
Wireless Toolkit 2.2 [J2ME Wireless Toolkit]

Eclipse 3.x and Wireless Toolkit 2.x

1. Install and start Eclipse 3.0 / Eclipse 3.1
2. Configure Eclipse for J2ME:
In the Eclipsemenu: "Help", "Software Updates", "Find and Install", "Search for new features to install" and press "Next".
Press "New archived site" and select the file "eclipseme.feature_0.6.1_site.zip".

This tool is available on `http://eclipseme.sourceforge.net/`

Select and install this tool in Eclipse.

In the Eclipsemenu: "Window", "Preferences", "J2ME", "Platform Components", rightclick on "Wireless Toolkits" and click "Add Wireless Toolkit", Browse to the folder of the Wireless Toolkit 2.2 and press "Ok".

3. Import the source code in a j2me-project, package "midlet".
4. Import the libraries. rightclick on the project, "Properties", "Java Build Path", "Libraries", "Add external JARs.." - Add the library kXML-RPC_full.zip

kXML-RPC:
`http://kxmlrpc.objectweb.org`
`http://kxmlrpc.objectweb.org/software/downloads/current/kxmlrpc_full.zip`

If you have problems with that - be sure that the classpath is linked to a full jdk-installation. In the Eclipsemenu: "Window", "Preferences", "Java", "Installed JREs"; Look for location of J2RE! Press edit, and browse to a full JDK-installation like "j2sdk_1.4.2".

5. Import the images: The images are available on CVS.
"../sensation/Pictures/MobileClient"

Put them into folder "../workspace/projectname/verified/classes/".

Now compile the project as J2ME-project. Mainclass is "MidletMain.java". Check, if profile is MIDP 1.0.

For more information about the installation, please check

`http://eclipseme.sourceforge.net/docs/installation.html`

Wireless Toolkit 2.x

1. Install the Wireless Toolkit 2.x
2. Start the jad-file "Mobile.jad". The only requirement is, that the jar-file "Mobile.jar" exist in the folder of the jad-file!

Emulators for Mobile Devices

This program is optimized for the mobile-phone Nokia 6230. To use this program in an emulator for this type, please download the emulator (S40 DP 2.0 SDK) at:

`http://www.forum.nokia.com/main.html`
`http://www.forum.nokia.com/main/0,,016-2062,00.html?model=6230`

To start the program, press "File" in the menu and then press "Open..". Now browse to the folder of the jar-file "Mobile.jar" select the jar-file. Then press "Ok".

9.6.3. Description of the MIDlet

Overview

An important aspect of the project was it to implement different clients. This documentation describes a program, particularly for mobile devices. The mobile devices have some special characteristics. Mobile devices have limited resources (memory, cpu). At least a 16 or 32 bit CPU, 160 kB memory and 32 kB runtime memory. Normally a mobile device has only a small display (128 x 96 px). Each mobile device represents the graphical user-interface in a different way. This program is adapted for the mobile phone - Nokia 6230. Normally the program fits on every other mobile device, but the representation of graphics etc. can be different because of the display-size. Another characteristic is the way to communicate with the server. Because of the high costs, it is not meaningful to have a constant connection. Mobile devices also have no IP-address, so they cannot be simply informed when an event occurs. At present it is only meaningful, to request the sensor data in an active way. Another characteristic is the restriction in programming the MIDlet. Because of the limited resources, the intention for the programmer is to keep the program as small as possible. Therefore some functions of the libraries, even whole libraries of java were completely omitted. In the following section I will describe the individual important components of the program.

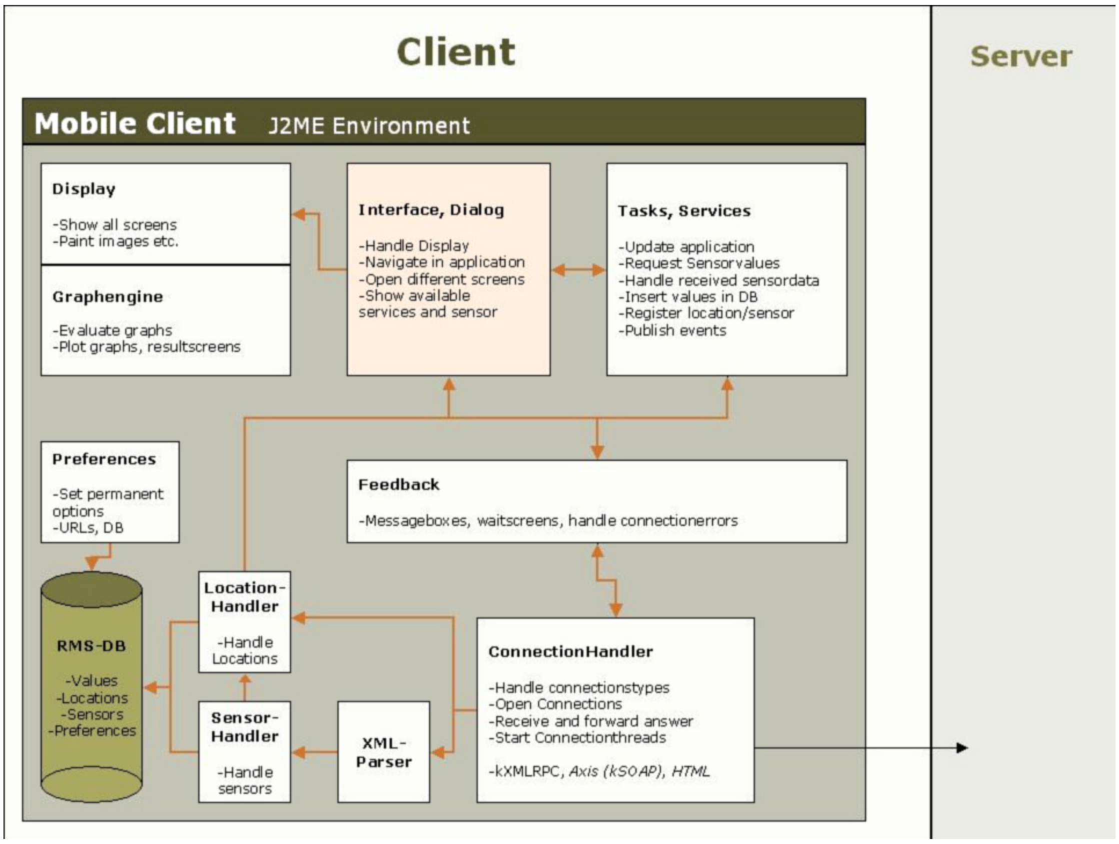


Figure 9.4.: Infrastructure of the mobile client.

Main Class

MidletMain.java

A MIDlet is a Mobile Information Device Profile (MIDP) application. A MIDlet's main class defines three life-cycle notification methods: `startApp()`, `pauseApp()`, and `destroyApp()`. In the active state the MIDlet is active. In the paused state the MIDlet instance has been constructed and is inactive. In the destroyed state the MIDlet has been terminated and is ready for reclamation by the garbage collector. The MIDlet has an Display-object, which displays the different screens. The MIDlet has to manage its screens and switch between them. A screen can be a Form (display different items like ChoiceGroups, TextFields etc.), a List, a Canvas (paint on display, show images), a TextBox or an Alert. When the program is started, the code in the method `startApp()` is executed.
The MIDlet provides a Display-object "display" and displays the Canvas "main"

on the display. This Canvas fills the background white and shows the logo of the project.
The constructor of this class loads the background-image, adds the commands and sets the CommandListener. Each screen which reacts to user-inputs needs a CommandListener or an ItemStateListener. What to do, when user-input happens is implemented in the method `commandAction()`.
On the main screen, the user can click the Menu-button or the Quit-button. If clicking the Menu-button, the next screen (`ProgramMenu`) will be opened. If the user clicks the Quit-Button, the method `destroyApp()` will be called and the program terminates. The method `destroyApp()` defines, what to do when the program ends. In this case, the LocationStore and the UpdateStore will be deleted, to configure the program for the next use.
The Main class also displays every MessageBox. MessageBoxes are important to inform the user about any errors which occurs, or to give any other feedback. The method `message()` paints the MessageBox on the display. The contents of the MessageBox are drawn in the class `Messages`.
The method `displayScreen()` can switch the displays. The most classes in this program are even screens (Menus etc.). This method gets an integer value and decides, which screen will be displayed. When this method is started, the constructor of the class which will be opened adds the implemented items to the screen.

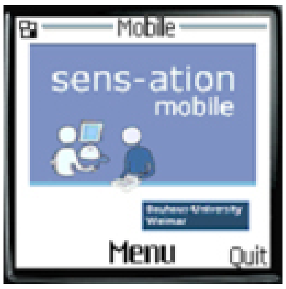


Figure 9.5.: Startscreen of the program.

Main Menu

ProgramMenu.java

The `ProgramMenu` is the main menu in this program. This class extends the class List. That means, this class is a screen of the type List. This screen is

displayed via the Display-object in the class `MidletMain`. Before doing this, a new list will be created. Afterwards the constructor inserts all items in the List and sets the commands and the CommandListener.
The user can choose between 5 different list-items.

1. Sensormenu

2. Servicemenu

3. Preferences

4. Registrationmenu

5. Publish event

When the user has choosen a list-item and he clicks the select-button, the method `commandAction()` starts. This method gets the index of the selected item and executes the code for this item. Before starting the `Sensormenu` (index 0) or the `Servicemenu` (index 1), the program examines whether the program is updated. (The method `isUpdated()` in the class `LocationHandler` returns **true** if yes and **false** if no.) If the program is updated, the menu will be opened, otherwise the `UpdateScreen` will be opened. This guarantees that the program is updated while it is used. When the user clicks "Publish event" it is also recommend, that the program is updated.

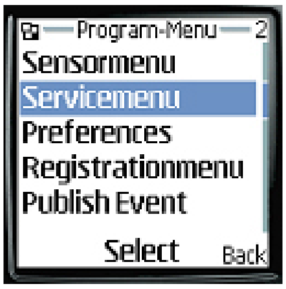
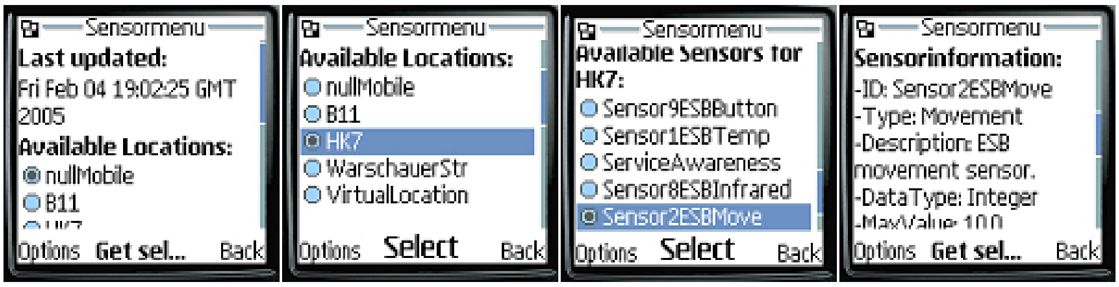


Figure 9.6.: The Programmenu.

Sensormenu and Servicemenu

SensorMenu.java, ServiceMenu.java

In this menus, the user can choose a sensor of a certain location. Then he can start a task, like requesting the sensor values of this sensor or requesting the values of a chosen service. The first step is, to choose a location from the the ChoiceGroup which contains all available locations. After this all available sensors for this locations are shown. Then he only need to choose one of the sensors. When clicking a sensor, some important sensor information is shown in a separate TextBox. The user also can update the program once again. All this functions are listed in a menu within this form.
From the aspect of programming, the `SensorMenu` and the `ServiceMenu` are the same. Both classes extends the class `Form`. I will describe, how the classes work at the example `SensorMenu`.
The Form contains two ChoiceGroups, one for the locations and one for the sensors. In addition the Form contains some StringItems, for display text in the Form. For example the last update in form of a Date-object. The constructor in-


Sensormenu
Last updated:
Fri Feb 04 19:02:25 GMT 2005
Available Locations:
nullMobile
B11

Sensormenu
Available Locations:
nullMobile
B11
HK7
WarschauerStr
VirtualLocation

Sensormenu
Available Sensors for HK7:
Sensor9ESBButton
Sensor1ESBTemp
ServiceAwareness
Sensor8ESBInfrared
Sensor2ESBMove

Sensormenu
Sensorinformation:
-ID: Sensor2ESBMove
-Type: Movement
-Description: ESB movement sensor.
-DataType: Integer

Figure 9.7.: 1. The last update, 2. The available locations, 3. The available sensors for this location, 4. The sensorinfo.

serts the items in the form. The StringItems "noLocationAlert", "noSensorAlert" or "noSensorInfoAlert" are empty but they are also inserted into the Form. For example, if there is no Sensor available, the StringItem "noSensorAlert" gets the Text "No Sensor available". That means, this StringItems only have the function to inform the user.
Normally the program is already updated, when the user can start the `SensorMenu`. The contructor examines, when the program was updated (The method lastUpdated in the class `LocationHandler` returns a String with the last update) and sets the date of the last update in the StringItem "lastUpdate". Otherwise it gets the text "never updated". The StringItem "sensorInfo" contains the description of the chosen sensor. Then the constructor inserts the StringItems and the

ChoiceGroups in the Form. The contents of the ChoiceGroups were inserted in the update-process of the program. Also the commands for the different tasks and the commandListener will be added to the Form.
The most methods of this class will be used in the update process. I will describe them in section ”Update program”.
Another important method for this screen is the method `itemStateChanged(Item item)`. This method will be activated, if a user clicks on an item in the ChoiceGroup. When this menu is opened the first time, the user only see the available locations in the location-ChoiceGroup. When he clicks on a location (Select-button), the method `itemStateChanged(Item item)` starts. In this case, the parameter item is the ChoiceGroup ”locations”.
Depending on the selected location, the method searches the sensors, which are available for this location, in the sensorStore. The sensors are saved as String in the sensorStore in a special way:

For example a sensorRecord:

```
"HK7|ID|Sensor1|Description|This is a sensor|..."
```

This String can be parsed with the method `tokenise()` in the class`Utility`. This method returns an array and each splittet element of this String is an element in the array. The first element is the name of the location (HK7), the next important element is the third one (Sensor1). This is the name of the sensor.
This name will be inserted in the ChoiceGroup ”locations”. The method `itemStateChanged(Item item)` searches all sensors of the selected location via a while-loop and inserts them. If there is no sensor available, the StringItem ”noSensorAlert” gets the text ”No Sensors available”.
Now the user can select a sensor on the screen. If he selects a sensor, the method searches in the ”sensorStore” for the selected Sensor and fills the StringItem with the sensor information. How is it done? The method scans the record. The String is parsed to an array. The method scans the array for keywords like ”ID” or ”Description” and if it finds such a keyword, the next item in this array contains the value of this keyword.
For Example:

String in sensorRecord:

```
"HK7|ID|Sensor1|Description|This is a sensor|NativeDataType|Integer|..."
```

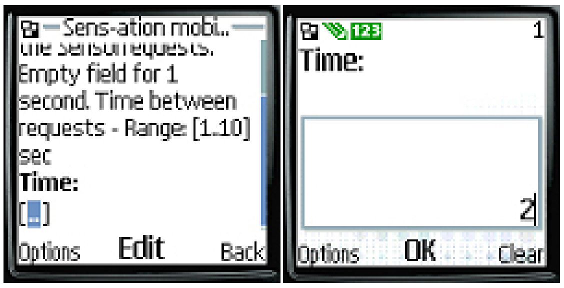
Content of the Array after the String was parsed:

```
{HK7,ID,Sensor1,Description,This is a sensor,NativeDataType,Integer,...}
```

Formatted text of the StringItem:

```
LocationID: HK7
SensorID: Sensor1
Description: This is a sensor
DataType: Integer
.
.
```

The String in the records is a parsed String of the classe `XMLParser`. They will be set as records after the update process.
In the `SensorMenu` and the `ServiceMenu`, the user can start the main tasks of the program like requesting sensor values or values of services. When the user has selected a sensor and he chooses a task, the method `commandAction()` will be activated.
The user can update the program once again. This function is described in the section "Update Program".
Another Task is "Get selected". In this case, the method finds out the type of the selected Sensor with the method `getSensorType()`. This method scans the sensor information for the entry "NativeDataType". Afterwards, thie minimum value and the maximum value of the sensor will be scanned in the same way with the method `setSensorInfo("MaximumValue")`. The argument for this method is the requested information. After doing this, the database which saves the sensor values is prepared and the screen `OptionsMenu` opens. In this screen, the user sets the time between each sensor request. When clicking starting the Ok-Command


Sens-ation mobi...
the sensorrequests.
Empty field for 1 second. Time between requests - Range: [1..10] sec
Time:
[...]

Time:
2

Figure 9.8.: In this screen, the time is set.

the CommandListener of this class opens the `WaitScreen` (described in another

section), examines the inserted time and starts the connection by opening the method `task()` in the class `ConnectionHandler`. The method gets the arguments "singleSensor" as task (Important for the class `PrepareTasks`), "getValueString" as called method on the server, null as locationID, the sensorID of the selected sensor and the inserted time.
If the User selects the task "Get Awareness", the connection starts directly from this class with the task "multipleSensorsRealtime" (Important for the class `PrepareTasks`). In this case, the program requests only three sensors, which are important for the awareness. (This task is not very meaningful, because this task depends on the location and on the embedded sensorboard.)

9.6.4. Handling of different Connections

ConnectionHandler.java

This class handles all outgoing connections. The mainly kind of Connection is XML-RPC. When a program wants to start a connection, one of the methods...

```
public void task(      String task,
                       String method,
                       String locationID,
                       String sensorID,
                       int time)

public void register(  String method,
                       String xmlString)

public void publish(   String method,
                       String sensorID,
                       String dateStamp,
                       String event)

public void login(     String method,
                       String login,
                       String password)
```

...is called. All methods start a separate thread, which executes the connection. It is important to do the connection not with the system thread, because of errors while doing the connection. This prevents system crashes etc. In addition other tasks like the `Waitscreen` can use the system thread. The method `task(..)` is called, to request any sensor- or service data. Before the program connects to the server to request such data, the request has to prepared in the Class `PrepareTasks`. This class calls the method `makeRequest(..)` in the

`ConnectionHandler`, which starts the connection with the prepared arguments. The other methods starts their own threads. The instructions for the threads are implemented in the `run()`-methods of the threadclasses.
Each connection needs an instance of the class `XmlRpcClient`. The given argument is the URL of the server.

```
private XmlRpcClient xmlrpc;
xmlrpc = new XmlRpcClient("http://141.54.159.128:5000");
```

XML-RPC calls a method on the server and waits for the answer. The method is given in the argument "method". Arguments for the called methods on the server will be packed in a vector.

```
Vector params = new Vector();
params.addElement("string-argument");
```

A variable gets the answer of the server. A request will be executed in this way:

```
tempVariable = "" + xmlrpc.execute("SensorPort." + method, params);
```

"SensorPort." is the class on the server, method is the given method and the vector contains the arguments. Every request to the server is executed in this way. Only the kind of request is different. For example the `login(..)`-request has the arguments "String name, String password" and gets a String with the available sensors and services for this account as answer. When login failed, the answer is an empty String.
The method `makeRequest(..)` opens the connections for the method `task(..)`. The job of this method is it, to differentiate the kind of the requests (update, request sensor- or service values, request available sensors or available locations). One additional task when requesting a sensorValue, is to prevent errors with errors from the sensor. The answer from the server is scanned for errors (errors contains chars like '[' or ']') or if the answer is float, the answer will be converted in an integer value by cutting the float value. This class only starts connections and return the answers of the server.

Preparing the Requests

PrepareTasks.java

This class prepares different tasks and starts the connections. The main method in this class is:

```
sensorTasks(  String task,
              final String request,
              final String locationID,
              final String sensorID,
              int time)
```

Dependently on the first argument "task", it prepares the different tasks.

One task is "getLocations":

In this task, the locations will be requested. The main part of this task is to handle the answer and insert the locations in the locationStore and into the ChoiceGroups of the `SensorMenu` and the `ServiceMenu`.
If the answer of the server is not null, the method `updateLocationList(answer)` with the answer of the server as argument is called. This method inserts the available locations in the locationStore. This method also requests all sensors of every available location and inserts them into the sensorStore. Then the method `fillLocationList()` in the `SensorMenu` and the `ServiceMenu` is started. This method inserts all locations in the ChoiceGroups of this forms, so they are shown as items in the ChoiceGroups when the user navigates once again to the `SensorMenu` or the `ServiceMenu`. The Task "getSensors" is similar to the task "getLocations".

Another Task is "stringSensor":

This task is opened, when a user requests a sensorvalue, and the native datatype of this sensor is String. Normally the values are numeric and they will be shown in a chart-diagram. A String will be shown in a TextBox. If so, the task requests this String from the server and opens the class `ShowStringSensor`. This class extends the TextBox and inserts the text into this TextBox. This looks like getting a SMS.

Another Task is "singleSensor":

This task is the main task for requesting sensor values. This task starts a Timer and starts sensor requests in the time intervals, which were set from the user in the `OptionsMenu`. Every request saves the answer from the server in the `ValueDataBase`. After receiving the first answer from the server, the task starts the class `Graph`, which displays the gotten sensor values. The task terminates when the user clicks the Ok-button in the graphic-screen. If this occurs, the method `terminateConnection()` starts and sets the boolean variable "quit" to true. The task examines the state of this variable before each request. If it is true, the Timer cancels and the `SensorMenu` opens.

The other Tasks "multipleSensorsRealtime" (Get Awareness) and "startService" are similar. They also request sensor values, save them in the `ValueDataBase` and starts the Graphmanager. This tasks are also realized by using a Timer. The difference is the way, how the sensor values are visualized and the kind of the requested values.

Feedback and Messageboxes

WaitScreen.java, Messages.java

This classes inform the user and give feedback, if something happens while he uses the program. The class `WaitScreen` visualizes an Animation, while a connection is active. This is important, because a user do not know, if a connection is active or not. In addition he do not knows, how long it lasts. In the worst case the user starts several connection, because nothing happens on the screen while the connection is active. The class `Messages` visualizes MessageBoxes on the screen for some seconds, to inform the user if a connection failed or if a task is ready etc.

WaitScreen.java

This class extends the Canvas-class. The constructor of this class loads the needed images for the animation and starts the TimerTask. This Timer loops every 200 ms and changes the represented image and repaints the Canvas to show the new image.
This animation ends, when the user clicks the Cancel-Command or the connection was successful. If the user clicks the Cancel-Button, the connection will be terminated by starting the method `terminateConnection()` in the class `PrepareTasks` and the `ProgramMenu` will be opened. If the connection was

successful, the `WaitScreen` ends in three different ways.

1. When the program was updated, the update continues and the appropriate screen will be displayed.

2. When the connection was a sensor request and the sensor had the type Integer or Float, the `WaitScreen` ends and starts the Graphicsengine with appropriate type.

3. When the connection was a sensor request and the sensor had the type String, the `WaitScreen` ends and starts the class `ShowStringSensor`.

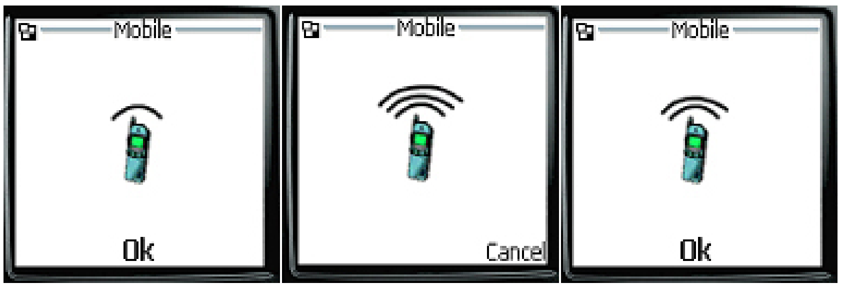


Figure 9.9.: The animated waitscreen.

Messages.java

Similiar to the `WaitScreen`, the constructor of this class loads the images (Stopsymbol, Infosymbol), starts the Timer and repaints the screen to show the MessageBox. The Timer is needed, to show the the MessageBox for a certain time.
Before a task opens a MessageBox, it sets the needed parameters with the method `setVariables(..)`.

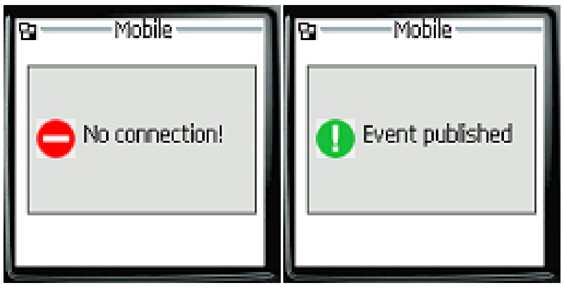
```
setVariables( String messageText ,
              int time ,
              int screen ,
              String type)
```

1. String messageText: The infotext, shown in the MessageBox

2. int time: The time, how long the MessageBox will be shown

3. int screen: The screen, which will be opend after showing the MessageBox

4. string type: The type of the MessageBox (Error, Info)



No connection!

Event published

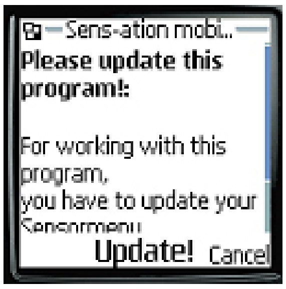
Figure 9.10.: Errormessage and Infomessage.

Procedure of updating the MIDlet

In this part, I will describe the procedure of updating the program. In the first step, the method `task(..)` in the class `ConnectionHandler` starts a new Thread. The method `run()` of this Thread starts the method `sensorTasks(..)` in the class `PrepareTasks` with the task "getLocations". This method requests (`ConnectionHandler.makeRequest(..)`) the available locations from the server. The server returns a String, which contains all available locations. For example:

```
"HK7|B11|Virtual Location|Warschauer Strasse|.."
```

If this String is not null, the date of the last update in the`SensorMenu` and the `ServiceMenu` will be renewed. The method `updateLocationList(..)` in the class `SensorMenu` fills all locations in the locationStore (`LocationHandler.updateLocations(..)`; This method fills the locations in the ChoiceGroup of the `SensorMenu` and the `ServiceMenu`), and starts a while-loop which requests the available sensors for every location in the locationStore in the same way. In this case the server returns a XML-String, which contains all sensors for the selected location and the whole sensor-description. This String will be parsed in the class `XmlParser`. The main method `parse(String xmlString)` returns a parsed String. The requested sensors will be filled in the sensorStore by using the method `updateSensors(..)` in the class `SensorHandler`. This method splits the string and puts all sensors with all information as records in the recordstore. If a sensor has type "other" it will be handled as service. In the last step, the `Waitscreen` ends and opens the appropriate menu.


Sens-ation mobi...
Please update this program!:

For working with this program,
you have to update your Sensormenu
Update! Cancel

Figure 9.11.: This program informs the user about a needed update.

Registration of Locations and Sensors

RegisterLocation.java, RegisterSensor.java

Both classes extends the class Form and both classes are nearly similar to each other. The constructor inserts the Commands and the Textfields into the Form.

```
.
.
ensorID = new TextField("SensorID", "", 100, TextField.ANY);
    append(sensorID);

sensorClass = new TextField("SensorClass", "", 100, TextField.ANY);
    append(sensorClass);

description = new TextField("Description", "", 100, TextField.ANY);
    append(description);

hardwareID = new TextField("HardwareID", "", 100, TextField.ANY);
    append(hardwareID);

command = new TextField("Command", "", 100, TextField.ANY);
    append(command);
.
.
addCommand(REGISTER_CMD);

addCommand(BACK_CMD);

setCommandListener(this);
```

When the user inserts the properties and when he clicks ”Register”, the CommandListener produces a XML String from the properties out of the TextFields.

```
"<Sensor id="sensor1" class="Temperature">
 <Description>This is a sensor</Description>
```

```
<HardwareID>...</HardwareID>
<Command>...</Command>
.
.
</Sensor>"
```

Afterwards the CommandListener starts the method `register(..)` (with the XML String as argument) in the `ConnectionHandler`. This method starts the connection and registeres the sensor or the location. When the connection was successful, the program shows a MessageBox (info- or errormessage).

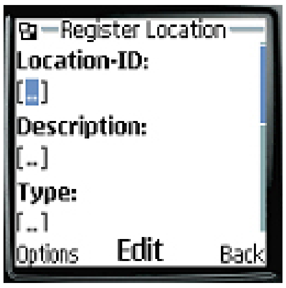

Register Location
Location-ID:
[..]
Description:
[..]
Type:
[..]

Figure 9.12.: The Registerscreen.

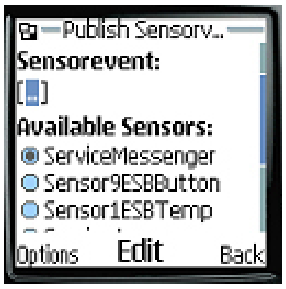
Publication of Sensor Events

PublishSensorValue.java

This class also extends the class Form. The user navigates to this screen from the `ProgramMenu`. Before the CommandListener of the `ProgramMenu` opens the Publish-screen, it starts the method `updateSensors()` in the class `PublishSensorValue`. This method inserts all sensor from the sensorStore into the ChoiceGroup of this Form. When the screen opens, the constructer displays the ChoiceGroup with all available sensors and a TextField, which is used to insert the sensor event. The user can select a sensor and insert a value for this sensor in the TextField. When he clicks the "Publish-button", the CommandListener starts the method `publish(..)` in the `ConnectionHandler`. This method starts the connection and publishes the sensorvalue for the selected sensor.

```
connectionHandler.publish(  "notify",
                            sensor,
                            dateStamp,
                            event);
```

The first argument is the called method on the server. The second argument is the name of the selected sensor. The third argument is a generated datestamp as String and the last argument ist the sensorvalue of this event.
When the connection was successful, the program shows a MessageBox (info- or errormessage).



Sensorevent:

[..]

Available Sensors:

● ServiceMessenger
○ Sensor9ESBButton
○ Sensor1ESBTemp

Figure 9.13.: The Publishscreen.

The Graphicsengine

Graph.java

This class extends the class Canvas. This class is called after or during a sensor- or service request. When the `WaitScreen` ends in a graph, it calls the Graphicsengine. Before a graph is displayed, the `WaitScreen` sets some properties. The method `setDiagramType(String type)` sets the type of the diagram (service awareness, simple sensor values, multiple sensor values). The method `setMinMax(int minValue, int maxValue)` sets the minimum- and the maximumvalue of the selected sensor (Taken from the sensor description). The method `setSensorInfo(String sensorName)` sets the name of the selected sensor. The method `setPosition(int pos)` is needed, to synchronize the repainting of the Graphicsengine with the sensor requests.
While the Graphicsengine displays this sensor values (taken from the valueStore - `ValueDataBase`), the certain task request sensor values and fills the valueStore with new values. The Graphicsengine repaints the Graph every few seconds and gets all new values. The services will be repaintet without a pause. This will be done until the user clicks the Ok-button.
I will describe one diagram in detail. The others are described in the sourcecode.

Diagram 1 - simple sensor

Split the display in fields.

```
int x1 = getWidth()/11;
int y1 = getHeight()/13;
```

Paint the whole background darkgrey and the background of the chart in lighter gray.

```
g.setColor(160, 160, 160);
g.fillRect(0, 0, getWidth(), getHeight());

g.setColor(220, 220, 220);
g.fillRect(x1, y1*2, x1*(10), y1*(10));
```

Now the requested sensorvalue gets a percentual value dependently to the maximumvalue of the sensor. Remember: J2ME do not supports Float. Maybe the percentual Value is not exact.

```
worth = (valueDataBase.getValue(1)*100) / max;
```

Now the color of the filled bar is setted, dependently of the sensorvalue.

```
g.setColor(41*worth/100, 186*worth/100, 253*worth/100);
```

Afterwards the bar and a lighter grey outline will be paintet, depending on its position.

```
g.fillRect( i*x1,
            (12*y1)-(((10*y1)*worth)/100),
            x1,
            ((10*y1)*worth)/100);
g.setColor(240, 240, 240);
g.drawRect( i*x1+1,
            (12*y1)-(((10*y1)*worth)/100),
            x1-2,
            ((10*y1)*worth)/100);
```

Then the inscription will be paintet, the present position in orange.

```
g.setColor(0, 0, 0);
```

```
g.drawString( String.valueOf(i),
              i*x1+2+1,
              (12*y1)+1,
              Graphics.TOP | Graphics.LEFT);
g.setColor(255, 255, 255);
g.drawString( String.valueOf(i),
              i*x1+2,
              (12*y1),
              Graphics.TOP | Graphics.LEFT);
g.setColor(239, 123, 32);
g.drawString( String.valueOf(position),
              position*x1+2,
              (12*y1),
              Graphics.TOP | Graphics.LEFT);
```

After this, the program paints the coordinates, the range of the sensor values and the name of the sensor.

```
for (int i=0; i < getWidth();i=i+x1) {
        g.drawLine(i,(2*y1),i,y1*12);
      }
for (int i=2*y1; i < y1*13;i=i+y1) {
        g.drawLine(0,i,getWidth(),i);
      }
g.setColor(255, 255, 255);
g.drawString( String.valueOf(max),
              1,
              (2*y1),
              Graphics.TOP | Graphics.LEFT);
g.drawString( String.valueOf(min),
              1,
              (11*y1),
              Graphics.TOP | Graphics.LEFT);
g.setColor(0, 0, 0);
g.drawString( sensorName,
              2,
              2,
              Graphics.TOP | Graphics.LEFT);
g.setColor(255, 255, 255);
g.drawString( sensorName,
              1,
              1,
              Graphics.TOP | Graphics.LEFT);
```

This will be repaintet once again, when a new sensorvalue is present.

The XML-Parser

XML-Parser.java

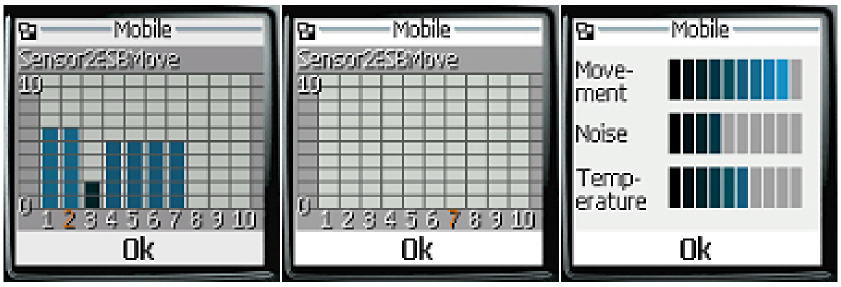


Figure 9.14.: The visualization of the given sensor values. 1. and 2. The simple sensor request, 3. Get awareness.

This class is a utility, used to parse the XML sensor descriptions from the server. The main method is `parse(String xmlString)`. This method gets such a XML String and returns a String with the parsed important elements.

XML String:

```xml
<Sensors>
  <Sensor id="ServiceMessenger" class="Service">
    <Description>Notification of an messenger presence event.</Description>
    <HardwareID />
    <Command />
    <LocationID>VirtualLocation</LocationID>
    <Owner>Cooperative Media Lab</Owner>
    <Comment>Notification occurs, when a messenger presence information is
    registered.</Comment>
    <AvailableSince>2005-01-01 10:00:00</AvailableSince>
    <AvailableUntil>2005-12-01 12:00:00</AvailableUntil>
    <SensorActivity activity="active" />
    <NativeDataType>String</NativeDataType>
    <MaximumValue>0.0</MaximumValue>
    <MinimumValue>0.0</MinimumValue>
  </Sensor>
</Sensors>
```

The parsed String:

```
ID|ServiceMessenger|Type|Service|Description|Notification of an messenger
    presence event.|NativeDataType|String|MaximumValue|0.0|MinimumValue|0.0|
```

`XmlParser` initializes an instance of ByteArrayInputStream with the XML-String as input. Then it initializes an instance of `XmlParser`.

```java
ByteArrayInputStream bin = new ByteArrayInputStream(xml.getBytes());
XmlParser parser = new XmlParser( new InputStreamReader( bin ) );
ParseEvent event = parser.read ();
```

This Stream will be parsed until the end.
A XML-String consists of start-tags, end-tags, whitespaces, text and some other components. For example: When a start-tag ”Sensor” is determined, the parser takes the name of the sensor and inserts it in the output-string. This output-String is prepared for the tokeniser in the class `Utility`.

```
title = "ID|" + event.getAttribute("id").getValue() + "|";
output = output + title;
```

When the whole XML-String is parsed, the method returns the output-String.

The library for the `XmlParser` is contained in the XmlRpcLibrary. [kXML-RPC]

Handling of requested sensors, locations and sensor values

Sometimes the program has to store data like sensors and services, locations, sensor values, programsettings etc. A MIDlet cannot write any files to store date permanent. The only possibility to store data in a permanent way are RecordStores. RecordStores are created in platform-dependent locations, which are not exposed to the MIDlets. The naming space for RecordStores is controlled at the MIDlet suite granularity. MIDlets are allowed to create multiple record stores, as long as they have different names. When a MIDlet is removed from a platform all the record stores associated with its MIDlets will also be removed.
Records are identified within a given RecordStore by their recordId, which is an integer value. This recordId is used as the primary key for the records. The first record created in a RecordStore will have the recordId (1). Each subsequent record added to a RecordStore will be assigned a recordId one greater than the record added before it.

ValueDataBase, LocalDataBase, PreferencesDataBase: LocalDataBase.java, ValueDataBase.java, PreferencesDataBase.java

All classes provides methods to open the stores (This builds a new instance of the RecordStore, also if the RecordStore is already opened.), methods to close the stores (This closes one instance this RecordStore) and methods to delete the stores (A RecordStore cannot be deleted, while there is an instance of this RecordStore opened.).

LocalDataBase:

This class handles a RecordeStore, to store some values which are used, when no connection is available. When a program starts first time, this stores have to be initialized in the `PreferencesMenu`. When the method `initStores()` is called, 7 RecordStores will be initialized and filled with a Record (value = 0) for every hour within this day.
When a user wants to get an average value for a certain time (Monday, 7 pm), the method `getValueGraph(int store, int index)` is called with the day (1 for Monday, .., 7 for Sunday) and the hour (7 pm -¿ 19) as second argument is called. This method returns the stored average value for awareness for this time.
The method setValue sets the average value in this RecordStore. In detail: "count" contains the number of inserts in this record (needed to solve the average value), "wert" contains the present average value, "newWert" contains the value of the present sensor request, "newCount" is the new number of all requests at this time, "newGesWert" gets the solved new average value for this hour. This value and the new "count"-variable will be inserted in the RecordStore.

```
int count = Integer.parseInt(temp[0]);
int wert = Integer.parseInt(temp[1]);
int newWert = Integer.parseInt(value);
int newCount = count + 1;
int newGesWert = ((wert * count) + newWert) / (newCount);
```

ValueDataBase

This class handles a RecordeStore, to store values after a sensor- or service request. The methods are very simple. The method `insertValue(String value)` inserts the given value at the end of the RecordStore. The method `setValue(String value, int recordId)` inserts a value at the position of an existing value and the method `getValue(int recordId)` returns the value of a record in a certain position.

PreferencesDataBase

This class handles a RecordeStore, which stores some preferences of the program like URLs etc. The methods are also the same as in the previous classes.

SensorHandler, LocationHandler, ServiceHandler: SensorHandler.java, LocationHandler.java, ServiceHandler.java

The function of both classes is nearly the same. This classes also got methods to open, close and delete their RecordStores.

SensorHandler, ServiceHandler

This classes handles a RecordeStore, which stores the available sensors after the program was updated. The method `updateSensors(String locationID, String objects)` opens the RecordStores, parses the given objects-String and inserts every sensor in an own record. The String objects contains the sensors in this way:

```
"|sensorID|value of sensorID|sensorClass|value of sensorClass|...|$|sensorID|
    value of sensorID|..|$|.."
```

First step is to split all sensors within this String. Second step is to split all sensor information within this subString. This is done with 2 loops. The information about the sensor are inserted in the RecordStore. The first element of the String is the name of the location, then the sensor information. If a sensor has the type "Other" it is a service and it will be inserted from the `ServiceHandler` in the same way.
The method `getSensor(int index)` returns the String of the sensor at the given position.

LocationHandler

This class does nearly the same. The method `updateLocations(String objects)` inserts the requested locations in the RecordStore. The method `getLocation(int index)` returns the String of the location at the given position. The method `lastUpdated()` returns a String which contains the date of the last update from the updateStore. At last, the method `isUpdated()` returns a boolean value, if the program is already updated. This method checks, if a record is in the updateStore.

10. Scenarios of using Sens-ation

In this chapter you can read two tutorials for working with the *Sens-ation* platform: adding new sensors or locations to the platform and the development of a new sensor adapter.

10.1. Adding new sensors and locations

1. Create a new java file `TestRegisterSensor.java` and import the package `org.apache.xmlrpc.*` (you need the Apache XML-RPC library for Java [Sens-ation Download])

2. Create the class constructor and initialize a new XML-RPC client object. You have to change the `server` string to specify the IP where the `Sens-ation` server is currently running.
   We also add two method calls for the register methods we create in the next step.

```java
public class TestRegisterSensor {
  /** The xmlrpc connection */
  static XmlRpcClient xmlrpc;

  /** Constructor */
  public TestRegisterSensor(){
    String server   = "http://localhost";
    String port     = "5000";

    try { xmlrpc = new XmlRpcClient(server + ":" + port); }
    catch (MalformedURLException e) { e.printStackTrace(); }

    this.registerLocation();
    this.registerSensor();
  }
}
```

3. First we create the `registerLocation()` method: First we add the XML

description string to the parameter vector (line 4) and try the submission (line 12).

```
1  public void registerLocation(){
2    Vector parameter = new Vector();
3    String locationID = "TheNewLocation";
4    parameter.add("<Location id=\"" + locationID + "\">"
5      + "<Description>The new location.</Description>"
6      + "<Type></Type>"
7      + "<DegreeOfLongitude>0.0</DegreeOfLongitude>"
8      + "<DegreeOfLatitude>0.0</DegreeOfLatitude>"
9      + "<HeightAboveSeaLevel>0.0</HeightAboveSeaLevel>"
10     + "</Location>");
11   try {
12     System.out.println(xmlrpc.execute("SensorPort.registerLocation",
       parameter));
13   } catch (XmlRpcException e) { e.printStackTrace();
14   } catch (IOException e) { e.printStackTrace(); }
15 }
```

4. Now we create the `registerSensor()` method: We use this method in an equivalent way as the `registerLocation()` method.

```
public void registerSensor(){
  Vector parameter = new Vector();
  String sensorID = "MyOwnSensor";
  parameter.add("<Sensor id=\"" + sensorID + "\" class=\"ASCII\">"
    + "<Description>Example Keyboard Sensor</Description>"
    + "<HardwareID></HardwareID>"
    + "<Command></Command>"
    + "<LocationID>HK7</LocationID>"
    + "<Owner>Your Name</Owner>"
    + "<Comment>The test sensor.</Comment>"
    + "<AvailableSince>2005-01-01 09:00:00</AvailableSince>"
    + "<AvailableUntil>2005-12-31 12:00:00</AvailableUntil>"
    + "<SensorActivity activity=\"active\" />"
    + "<NativeDataType>String</NativeDataType>"
    + "<MaximumValue></MaximumValue>"
    + "<MinimumValue></MinimumValue>" + "</Sensor>");
  try {
    String response = (String)xmlrpc.execute("SensorPort.registerSensor",
     parameter);
     System.out.println("The registered sensor: " + response);
  } catch (XmlRpcException e) { e.printStackTrace();
  } catch (IOException e) { e.printStackTrace(); }
}
```

5. All necessary methods are created, and we can write the main method:

```
public static void main(String args[]) throws XmlRpcException,
    IOException {
  TestRegisterSensor regLoc = new TestRegisterSensor();
}
  \item Now you can compile the sourcecode and run the program.
```

Note: Instead of using this software registration process, you can also add sensors and locations with the PHP admin interface (chapter 8.2) or you can add the description in XML format to the `sensors.xml` *or* `locations.xml` *file*

10.2. Development of a Software Sensor Adapter

1. Create a class `SimpleSensor` and import the package `org.apache.xmlrpc.*` (you need the Apache XML-RPC library for Java [Sens-ation Download])

2. Write the class file with the constructor: we need a private member of the type `XmlRpcClient` and initialize it in the constructor (change the IP if needed):

```java
public class SimpleSensor {
  /** The xmlrpc connection */
  static XmlRpcClient xmlrpc;

  /** Constructor */
  public SimpleSensor(){
    String server   = "http://localhost";
    String port     = "5000";

    try { xmlrpc = new XmlRpcClient(server + ":" + port); }
    catch (MalformedURLException e) { e.printStackTrace(); }
    System.out.println("Software sensor adapter started:");

    this.inputLoop();
  }
}
```

3. We use a `BufferedReader` instance to get text from the command line. Later you can write the event and click enter to send the message. For sending the text to the platform we need the `sendNotification()` method (line 14), that we implement in the next section.

```java
1    public void inputLoop(){
2      BufferedReader br = new BufferedReader(new InputStreamReader(System.
       in));
3      String inputLine = "";
4      boolean exit = false;
5      do {
6        System.out.print("Message :> ");
7        try {
8          inputLine = br.readLine();
9        } catch (IOException e) { e.printStackTrace(); }
10       if (inputLine == null) {
11         exit = true;
12       } else {
13         if(inputLine.equalsIgnoreCase("exit")) exit = true;
```

```
        else this.sendNotification(inputLine);
      }
    } while (!exit);
    System.out.println("Software sensor adapter exit.");
  }
```

4. The last method we need: the notification transmission to the platform server. In line 4 we set the sensor ID number, so you have to know that the sensor exists. If the sensor does not exists, the response of the `notify` method is false. To add this sensor, use the first tutorial of this chapter: "Adding new sensors and locations".

```
1   private boolean sendNotification(String event){
2     boolean success = false;
3     Vector parameter = new Vector();
4     parameter.add("MyOwnSensor");
5     parameter.add("");
6     parameter.add(event);
7     try {
8       Boolean response = (Boolean)xmlrpc.execute("SensorPort.notify",
    parameter);
9       success = response.booleanValue();
10    } catch (XmlRpcException e) { e.printStackTrace();
11    } catch (IOException e) { e.printStackTrace();
12    }
13    return success;
14  }
```

11. Limitations and Future Work

11.1. Known Bugs and Limitations

- **XML parsing:**
  If there will be invalid XML descriptions sent to the `SensorPort` or one of the sensor registry parts of the platform, there occurs unforeseeable errors. Due to the missing XML schema validation the XML parser tries always to read the XML description. There exists already a placeholder method for XML schema validation where the XML schema comparison can be inserted when finished.

- **Sensor Registry:**
  The sensor and service registry is only functions in the same subnetwork, that means the subnetmask must be the same. If not, the backing calls from the server will not be able to connect to the client. However, active calls from the client are still possible. Also critical is a change of IPs during having an active asynchronous service in use. You will have to register a new getting notified.

- **Database:**
  The database is able to save a big amount of data. Getting this data works fine. But you have could get problems, if you try to get more than 50000 sensor values from the database. You could get a buffer overflow. In the `Datamanager` we use this maximum amount, then we decompress the sensor values from the database. The amount of decompressed sensor values doesn't exceed this maximum amount.

- **Mobile Client:**
  The application is configured for the model Nokia 6230. For a correct representation it is recommened to use this model. Otherwise there can occur some errors because of the display size, particularly in the representation of

the graphics, menus and items.
If the application is updated all locations and sensors will be received within a simple String or a XML String. If this Strings do not correspond to the official format, there can occur problems like inserting the wrong elements into the RecordStores etc.
While a sensor request, the numerical values are transported as Strings. The most possible errors of the ESB are considered in the implementation, but if the given value of the sensor is corrupt in anyway and cannot be parsed, there can be occur an error.

- **XML-RPC Authentication:**
  The authentication will only work with XML-RPC 1.1 and the Apache XML-RPC 1.1 authentication patch (see chapter 2.4).

11.2. Future Work

- **XML schema:**
  An important part of the future work is the verification of the XML descriptions and modelling them in *XML schema*. With these descriptions it is possible to validate the descriptions before the parser tries to read the documents. XML schema is necessary for: sensors, sensor values, locations, sensor types and services.

- **Sensor adapter:**
  Many more sensors could be integrated in the sensor platform, for example some USB sensor boards, wireless connected sensors, connected hardware buttons (a control panel), a virtual USB camera sensor (for movement), "operating system" sensors (for keyboard and mouse activity) etc.
  It is just necessary to implement a further XML-RPC or sockets adapter and create the suitable sensor XML description. Perhaps the sensor XML format (see *XML schema*) must be modified to fit to the new sensor types.

- **Actors:**
  It would be interesting to create a concept for the integration of actors, e.g. LC displays, LEDs, speakers, high-voltage sockets (to control light sources) etc.
  A possible integration of actors could be the connection as client, that listens to a special kind of sensor (e.g. a service sensor, with intepretation) and can

activate or deactivate dependent on the sensor value. Another approach can be the modification of the hardware module implementations to handle the way back connections to the actors (sometimes integrated on the sensor board) hardware.

- **Services:**
The concept of services can be extended in various ways. It can be interesting to develop further services, to interpret the sensor values and aggregate different values to new calculated values. Another aspect of the further service development is to provide a dynamic service instantiation module that can for example create service class instances on the basis of a XML service description. In that case it would be helpful to create a GUI with a drag-and-drop like assembly interface.

- **WAP gateway:**
Instead of creating HTML websites for the mobile devices it could be useful to create WML WAP pages to enable the access to the server for more mobile devices (many mobile phones can not display HTML content but WAP pages). It is a challenge to use the advantages of WML created pages.

- **Security:**
The security is up to now only developed for XML-RPC connection. For the other gateways security is not implemented yet. The security against *Denial of Service* attacks is up to now only a beta. It consists merely of a function on the XML-RPC gateway that checks wether the request was made within a certain time period after the last call. There could be distinguished between trusted applications which are always allowed to fully access the gateway handler, and untrusted applications which are restricted in access.

- **Sensor registry:**
Up to now, all registration is done by sending the IP, the sensor and the port the client aplication is running on to the server. Thus, only via that way a notification can be done. For future implementation, registry and notification could be done not only in that way, but also via email or SMS. The way for notifying mobile cients has to be developed completely, because mobile clients can't be reached (up to now) by a static IP address.

- **Database compression:**
There is only one algorithm for compressing the database entries 7.3.2. But all sensor values are saved in one database table. So if the server runs for

days and saves sensor values in the database, you will get a very big database table. So it is worthwhile to create an alternate algorithm, for example to select the values from the last day or the last week and compress them.

- **Detection of sensor connections:**
  It would be helpful to implement a method that can determine the time a sensor was connected to the platform. It could be useful to specify a maximum time intervall for sensors to send a ”pulse” signal, so that the server module (and the database manager) can distinguish between the fact that the sensor only has no events to publish and the fact that the sensor is not connected to the platform anymore.

- **Mobile client visualization:**
  In future it is very important to handle the visulization of the service events in a more generic way. Because of the different kind of services it will be hard to consider the most favorable visualization for a given service event. The developers must find a efficiently way for the communication between the clients and the server about such problems.
  Another task is to find a simple and cognitive kind of visualization for service events. It is not meaningful to visualize service events in simple chart diagrams.

A. Appendix

A.1. XML Files

A.1.1. `sensors.xml`

```
<?xml version="1.0" encoding="UTF-8"?>

<!--     XML Sensor descriptions for the CML SENSOR INFRASTRUCTURE     -->
<!--  Copyright 2005 Cooperative Media Lab, Bauhaus University Weimar  -->

<!--  Note:                                                            -->
<!--  The sensor descriptions in this file will be parsed at server    -->
<!--  startup process. You can modify these descriptions, but you      -->
<!--  can also disable the sensor initialization: write '#' in front   -->
<!--  of the entry "server.sensorhandler.sensorfile" in the            -->
<!--  configuration file "server.properties".                          -->

<Sensors>
  <Sensor id="Sensor9ESBButton" class="Button">
    <Description>ESB hardware button; binary value.</Description>
    <HardwareID />
    <Command />
    <LocationID>HK7</LocationID>
    <Owner>Cooperative Media Lab</Owner>
    <Comment>This sensor is created via the XML description</Comment>
    <AvailableSince>2005-01-01 10:00:00</AvailableSince>
    <AvailableUntil>2005-12-01 12:00:00</AvailableUntil>
    <SensorActivity activity="active" />
    <NativeDataType>String</NativeDataType>
    <MaximumValue>0.0</MaximumValue>
    <MinimumValue>0.0</MinimumValue>
  </Sensor>
  <Sensor id="Sensor1ESBTemp" class="Temperature">
    <Description>ESB temperature sensor.</Description>
    <HardwareID />
    <Command />
    <LocationID>HK7</LocationID>
    <Owner>Cooperative Media Lab</Owner>
    <Comment>This sensor is created via the XML description</Comment>
    <AvailableSince>2005-01-01 10:00:00</AvailableSince>
    <AvailableUntil>2005-12-01 12:00:00</AvailableUntil>
    <SensorActivity activity="active" />
    <NativeDataType>Float</NativeDataType>
    <MaximumValue>30.0</MaximumValue>
    <MinimumValue>-20.0</MinimumValue>
```

```xml
  </Sensor>
  <Sensor id="MobilePhoneText" class="CellPhoneText">
    <Description>Mobile phone text information.</Description>
    <HardwareID />
    <Command />
    <LocationID>Mobile</LocationID>
    <Owner>Nicolai</Owner>
    <Comment>Test sensor to publish text information.</Comment>
    <AvailableSince>2005-01-01 10:00:00</AvailableSince>
    <AvailableUntil>2005-12-01 12:00:00</AvailableUntil>
    <SensorActivity activity="active" />
    <NativeDataType>String</NativeDataType>
    <MaximumValue>0.0</MaximumValue>
    <MinimumValue>0.0</MinimumValue>
  </Sensor>

  [...]

</Sensors>
```

A.1.2. services.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>

<!--    XML Service descriptions for the CML SENSOR INFRASTRUCTURE     -->
<!--  Copyright 2005 Cooperative Media Lab, Bauhaus University Weimar  -->

<!--  Note:                                                            -->
<!--  The service class descriptions in this file will be parsed at    -->
<!--  server startup process. You can modify these descriptions, but   -->
<!--  you can also disable the service initialization: write '#' in    -->
<!--  front of the entry "server.sensorhandler.servicefile" in the     -->
<!--  configuration file "server.properties".                          -->

<ServiceLoad>
  <ServiceClass>
    de.buw.medien.cscw.sensation.server.services.ServiceAwareness
  </ServiceClass>
  <ServiceClass>
    de.buw.medien.cscw.sensation.server.services.ServiceMessenger
  </ServiceClass>
</ServiceLoad>
```

A.1.3. locations.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>

<!--    XML Location descriptions for the CML SENSOR INFRASTRUCTURE    -->
<!--  Copyright 2005 Cooperative Media Lab, Bauhaus University Weimar  -->

<!--  Note:                                                            -->
<!--  The location descriptions in this file will be parsed at server  -->
```

```
<!--  startup process. You can modify these descriptions, but you can  -->
<!--  also disable the sensor initialization: write '#' in front       -->
<!--  of the entry "server.sensorhandler.locationfile" in the          -->
<!--  configuration file "server.properties".                          -->

<Locations>
  <Location id="Mobile">
    <Description>Mobile devices, e.g. cell phone or PDA. Can submit their
    location as event.</Description>
    <Type>inside</Type>
    <DegreeOfLongitude>0.0</DegreeOfLongitude>
    <DegreeOfLatitude>0.0</DegreeOfLatitude>
    <HeightAboveSeaLevel>0.0</HeightAboveSeaLevel>
  </Location>
  <Location id="HK7">
    <Description>Hauszknechtstrasse 7, 99423 Weimar. Bauhaus University.</
    Description>
    <Type>inside</Type>
    <DegreeOfLongitude>11.345</DegreeOfLongitude>
    <DegreeOfLatitude>51.91</DegreeOfLatitude>
    <HeightAboveSeaLevel>198.0</HeightAboveSeaLevel>
  </Location>
  <Location id="B11">
    <Description>Bauhausstrasse 11, Weimar. Bauhaus-University.</Description>
    <Type>inside</Type>
    <DegreeOfLongitude>11.305</DegreeOfLongitude>
    <DegreeOfLatitude>51.9</DegreeOfLatitude>
    <HeightAboveSeaLevel>200.0</HeightAboveSeaLevel>
  </Location>

  [...]

</Locations>
```

A.2. Properties Files

A.2.1. server.properties

```
# ----------------------------------------------------------------------
# SENS-ATION SENSOR INFRASTRUCTURE
# ----------------------------------------------------------------------
# COOPERATIVE MEDIA LAB, 2005
# SERVER PROPERTIES FILE
#
# Bauhaus University Weimar, Prof. Gross
#
# $Id: server.properties,v 1.8 2005/02/07 21:21:13 marquar1 Exp $
# ----------------------------------------------------------------------
# Notes: This property file contains the local setup preferences
#        for using the sensation infrastructure server, gateways
#        and adapter modules. For a developer doumentation and
#        more information about this project please visit
#        http://cml.medien.uni-weimar.de/
# ----------------------------------------------------------------------
# Property file created: Mon Dec 13 12:38:40 CET 2004, Nicolai Marquardt
# ----------------------------------------------------------------------



# PART A
# xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
# SERVER AND DATABASE CONFIGURATION
# MAIN CONFIG PART

# ----------------------------------------------------------------------
# Specify the database host address, the database name (by default you
# can let the name unchanged), user name and the password.
# ----------------------------------------------------------------------
database.hostIP=localhost
database.user=root
database.password=password
database.name=sensation

# ----------------------------------------------------------------------
# Use database: Values true or false
# ----------------------------------------------------------------------
server.usedatabase=false
# ----------------------------------------------------------------------
# Initialize locations and sensors from database: Values true or false
# ----------------------------------------------------------------------
server.initfromdatabase=false

# xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx





# PART B
# xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
# ADDITIONAL CONFIGURATION

# ----------------------------------------------------------------------
# Specifications of the database table names to save
# the sensor descriptions, values, users, subscriptions, etc.
# ----------------------------------------------------------------------
database.sensorValueTable=sensorValue
database.hardwareMetadataTable=hardwareMetadataTable
database.locationTable=locationTable
database.axisEventTable=axisEventTable
database.userTable=userTable
database.userSubscribeTable=userSubscribeTable
database.sensorTable=sensorTable
database.averageSensorDataTable=averageSensorValue
database.superuser=su
database.superuserpassword=su
database.sensorValueHistoryTable=sensorValueHistory
database.sensorHistoryTable=sensorHistory


# ----------------------------------------------------------------------
# Information about the server infrastructure and
# some parameters for server startup. If you have a local database
# or a valid remote connection to a database you can enable
# the 'usedatabase' property (set to value 1).
# ----------------------------------------------------------------------
```

```
server.description=Sens-ation server
server.location=YourLocation
server.owner=YourName
server.comment=Local test server for Sens-ation project

# ----------------------------------------------------------------------
# Use sockets: Values true or false
# Set to 'true' if the server has to start the socket
# listening class.
# ----------------------------------------------------------------------
server.usesockets=true

# ----------------------------------------------------------------------
# Authenticate: Values true or false
# Activates or deactivates the authentication check.
# If activated, the server needs for each XMLRPC call a valid
# user and password submission.
# ----------------------------------------------------------------------
server.authenticate=false

# ----------------------------------------------------------------------
# Initialize CML: Values true or false
# If set to 'true', the default example locations, sensors, etc.
# of the Cooperative Media Lab will be loaded.
# ----------------------------------------------------------------------
server.initializecmllocations=false
server.initializecmlsensors=false
server.initializecmlservices=false

# ----------------------------------------------------------------------
# Specifies the XMLRPC listeing port of the main server
# modules: GatewayHandler, SensorPort.
# ----------------------------------------------------------------------
server.xmlrpc.port=5000

# ----------------------------------------------------------------------
# Local XMLRPC Gateway: Values true or false
# If set to 'true', the xmlrpc gateway is started on the
# local server and not as remote module.
# ----------------------------------------------------------------------
server.xmlrpc.localgateway=true

# ----------------------------------------------------------------------
# Initialize the sensorTypes
# Specify the XML file that contains the sensorType descriptions
# ----------------------------------------------------------------------
server.sensorhandler.sensortypefile=sensortypes.xml

# ----------------------------------------------------------------------
# Initialize the sensors
# Specify the XML file that contains the sensor descriptions
# ----------------------------------------------------------------------
server.sensorhandler.sensorfile=sensors.xml

# ----------------------------------------------------------------------
# Initialize the locations
# Specify the XML file that contains the location descriptions
# ----------------------------------------------------------------------
server.sensorhandler.locationfile=locations.xml

# ----------------------------------------------------------------------
# Initialize the services (class files)
# Specify the XML file that contains the service descriptions
# ---------------------------------------------------------------------
server.sensorhandler.servicefile=services.xml



# GATEWAY PROPERTIES

# ---------------------------------------------------------------------
# Properties for the GatewayHandler. The local listener port is
# by default set to 5000, please only change this port if there
# is an important reason
# ---------------------------------------------------------------------
gateway.handler.timethreshold=500

# ---------------------------------------------------------------------
# Properties for the GatewayXMLRPC:
# Specifies the remote address and port of the GatewayHandler
# (server module).
# ---------------------------------------------------------------------
gateway.xmlrpc.port=5000
gateway.xmlrpc.ip=localhost

# ---------------------------------------------------------------------
# Properties for the GatewaySocket:
# Specify the listeing port (default is 6000) and the maximum number
# of clients connected to the server.
# ---------------------------------------------------------------------
gateway.socket.port=6000
gateway.socket.maxconnections=100

# ---------------------------------------------------------------------
# Specifies the default listening port of the notification
# service of new events. The server will contact clients
# through that specified port, except that they submit another port
# number.
# ---------------------------------------------------------------------
asynchronousclient.xmlrpc.ip=7000

# xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

A.2.2. gatewayxmlrpc.properties

```
# ---------------------------------------------------------------------
# Sensation Infrastructure
#
# Copyright Cooperative Media Lab, 2005
# Bauhaus University Weimar, Prof. Gross
# $Id: gatewayxmlrpc.properties,v 1.9 2005/02/03 12:16:12 pfaff Exp $
# ---------------------------------------------------------------------


# XMLRPC properties:
# this file contains the startup preferences for the xmlrpc gateway
# gatewayhandler: the adress of the gatewayhandler
# port: the port on which the gatewayhandler server runs
# authenticate: 0: no authentication, 1 : authentication
# timethreshold: the time in milliseconds in which multiple
# sensor accesses are fetched from the cache
# ----------------------------------------------------------------------
xmlrpc.localgateway=1
xmlrpc.port=5000
xmlrpc.gatewayhandler=localhost
xmlrpc.authenticate=0
xmlrpc.gatewayhandlerport=5000
xmlrpc.timethreshold=500
```

Bibliography

[Apache Ant] The Apache Ant Project: A Java-based build tool, http://ant.apache.org/, (website last visited: 12.02.2005) 15

[AppleScript Documentation] Apple Developer Connection, AppleScript Documentation
http://developer.apple.com/documentation/AppleScript/
(website last visited: 20.01.2005) 88, 90

[AppleScript XML-RPC and SOAP] Apple Developer Connection, XML-RPC and SOAP support
http://developer.apple.com/documentation/AppleScript/Conceptual/soapXMLRPC/index.html or the PDF file: http://developer.apple.com/documentation/AppleScript/Conceptual/soapXMLRPC/soapXMLRPC.pdf, (website last visited: 22.01.2005) 89

[Chen & Kotz 2002] Guanling Chen and David Kotz: Context Aggregation and Dissemination in Ubiquitous Computing Systems, Dartmouth College Science Technical Report TR2002-420, 2002 6, 51, 52

[Cooperative Media Lab Website] Cooperative Media Lab, Computer Supported Cooperative Work, Bauhaus University Weimar, http://cml.medien.uni-weimar.de/ 8

[Darwin 2001] Ian Darwin: Java Cookbook, First Edition, June 2001, Chapters 4 Pattern Matching and Regular Expressions, Chapter 11 Programming Serial and Parallel Ports 49

[Eclipse Foundation] Eclipse Foundation Website, Download and Documentation http://www.eclipse.org, (website last visited: 02.02.2005) 11, 54, 91

[Erich Gamma et al.] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, 1997 13

[ESB Documentation] Dokumentation of the Embedded Sensor Board, Freie Universitaet Berlin, http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb_net/ESB/sensorboards/doc/html/index.html, (website last visited: 05.01.2005) 48

[ESB Terminal Commands] C++ Terminal Dokumentation terminal.c, terminal.h http://www.inf.fu-berlin.de/inst/agtech/scatterweb_net/ESB/sensorboards/doc/html/terminal_8c.html, (website last visited: 05.01.2005) 49

[ISO 8601 Date and Time] ISO - International Organization for Standardization: Numeric representation of Dates and Time, http://www.iso.org/iso/en/prods-services/popstds/datesandtime.html, (website last visited: 29.01.2005) 32, 43, 83

[Java Code Conventions] Sun Microsystems, Code Conventions for the Java Programming Language, http://java.sun.com/docs/codeconv/ or the PDF file: http://java.sun.com/docs/codeconv/CodeConventions.pdf. (website last visited: 11.02.2005) 11

[Javadoc Reference] Sun Microsystems, Javadoc Reference: How to Write Doc Comments for the Javadoc Tool, http://java.sun.com/j2se/javadoc/writingdoccomments/ and http://java.sun.com/j2se/javadoc/index.jsp, (website last visited: 11.02.2005) 11, 15

[JDBC] The JDBC API
link:http://java.sun.com/products/jdbc/, (website last visited: 08.02.2005) 57

[J2ME Wireless Toolkit] J2ME Wireless Toolkit
A state-of-the-art toolbox for developing wireless applications.
http://java.sun.com/products/j2mewtoolkit/, (website last visited: 02.02.2005) 91

[kXML-RPC] kXML-RPC

A J2ME implementation of the XML-RPC protocol.
http://kxmlrpc.objectweb.org/, (website last visited: 28.01.2005) 112

[MySQL and Foreign Keys] MySQL reference book
http://dev.mysql.com/doc/mysql/en/ansi-diff-foreign-keys.html, (website last visited: 25.01.2005) 66

[MySQL Connector/J] MySQL Connector/J^{TM} The JDBC driver
http://dev.mysql.com/downloads/connector/j/3.0.html, (website last visited: 25.01.2005) 13, 57

[MySQL Database Website] MySQL^{TM} A open source database
link:http://www.mysql.com/downloads/index.html, (website last visited: 15.01.2005) 14, 57

[Sens-ation Download] Sens-ation donwload page,
Cooperative Media Lab, Computer Supported Cooperative Work, Bauhaus University Weimar, http://cml.medien.uni-weimar.de/tiki-index.php?page=Sens-ation-Download 21, 22, 27, 29, 115, 117

[Sens-ation Javadoc] Javadoc Pages of the Sens-ation project,
Cooperative Media Lab, Computer Supported Cooperative Work, Bauhaus University Weimar, http://cml.medien.uni-weimar.de/tiki-index.php?page=Sens-ation-Download 11