SEVENTH FRAMEWORK PROGRAMME

Specific Targeted Research Project

Call Identifier:	FP7–ICT–2011–7
Project Number:	287305
Project Acronym:	OpenIoT
Project Title:	Open source blueprint for large scale self-organizing cloud environments for IoT applications

D4.4.1 OpenIoT Integrated Development Environment a

Document Id:	OpenIoT-D441-131220-Draft
File Name:	OpenIoT-D441-131220-Draft.pdf
Document reference:	Deliverable 4.4.1
Version:	Draft
Editor(s):	Nikos Kefalakis, John Soldatos, Christos Georgoulis, Stavros Petris
Organisation:	AIT
Date:	2013 / 12 / 20
Document type:	Deliverable (Prototype)
Security:	PU (Public)

DOCUMENT HISTORY

Rev.	Author(s)	Organisation(s)	Date	Comments
V01	Nikos Kefalakis	AIT	2013/09/16	ToC proposal
V02	Christos Georgoulis	AIT	2013/11/08	Added chapters 3.1 & 3.5
V03	Nikos Kefalakis	AIT	2013/11/15	Corrections in chapters 3.1 & 3.5, added chapter 2
V04	A. Anagnostopoulos Panos Dimitropoulos	Sensap	2013/11/20	Added chapters 3.2 & 3.3
V05	John Soldatos	AIT	2013/11/25	Added chapters 1 & 4
V06	Prem Jayaraman	CSIRO	2013/11/29	Added content for chapter 3.4 (Sensor Schema Editor Interface)
V07	Nikos Kefalakis	AIT	2013/12/04	Corrections in chapters 3.2, 3.3 & 3.4. Ready for Technical Review
V08	Reinhard Herzog	FHG	2013/12/07	Technical Review
V09	Nikos Kefalakis	AIT	2013/12/10	Technical Review comments addressed. Ready for Quality Review
V10	Christian von der Weth	DERI	2013/12/11	Quality Review
V11	Nikos Kefalakis	AIT	2013/12/12	Quality Review comments addressed. Ready for Approval
V12	Martin Serrano	DERI	2013/12/15	Circulated for Approval
V13	Martin Serrano	DERI	2013/12/20	Approved
Draft	Martin Serrano	DERI	2013/12/23	EC Submitted

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TERMS AND ACRONYMS

Term	Meaning
AJAX	Asynchronous JavaScript and XML
AS	Application Server
CPU	Central Processing Unit
DoW	Description of Work
EJB	Enterprise Java Beans
GSN	Global Sensor Networks
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IERC	IoT European Research Cluster
IoT	Internet of Things
JDBC	Java Database Connectivity
JEE	Java Platform, Enterprise Edition
JSF	Java Server Faces
JSP	Java Server Pages
JVM	Java Virtual Machine
LGPL	Lesser General Public License
LSM	Linked Stream Middleware
OAMO	OpenIoT Application Model Object
OS	Operating System
OSMO	OpenIoT Service Model Object
POJO	Plain Old Java Object
RDF	Resource Description Format
REST	Representational State Transfer
SOA	Service Oriented Architecture
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language

SSN	Semantic Sensor Networks
UI	User Interface
UML	Unified Modelling Language
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WSN	Wireless Sensor Network
WYSIWYG	What You See Is What You Get
X-GSN	Extended Global Sensor Network
XML	eXtensible Markup Language

GLOSSARY AND TERMINOLOGY

Term	Meaning
(OpenIoT) Architecture	The set of software and middleware components of the OpenIoT platform, along with the main structuring principles and inter-relationships driving their integration in an IoT/cloud platform
(OpenIoT) Middleware	System level software (compliant to the OpenIoT architecture), which facilitates the integration of on-demand cloud-based IoT services.
(OpenIoT) Platform	A set of middleware libraries and tools, which enable the development and deployment of (OpenIoT compliant) cloud-based IoT services
(OpenIoT) Use Case	A goal or application serving needs of end users, which is implemented based on the OpenIoT platform.
(OpenIoT) Service	An IoT service deployed over the OpenIoT platform.
(OpenIoT) Scenario	A specific set of interactions between OpenIoT components serving the needs of an application
(OpenIoT) Cloud	A set of computing resources enabling the delivery of IoT services over the network and based on the use of the OpenIoT platform.
Global Scheduler	A software component that regulates how IoT services access the different resources managed by the OpenIoT platform.
Local Scheduler	A software component that regulates how IoT services access the local resources managed by an instance of the sensor middleware (and more specifically the GSN middleware).
Utility Metrics	A set of quantities/measures that are used for the metering of IoT services.
(OpenIoT) Service Delivery	The process of deploying and offering an OpenIoT service, after selecting the resources that are involved in the service
(OpenIoT) Request Presentation	The software components that visualizes the outcomes of an OpenIoT service based on the use of appropriate mashups and mashup libraries
Sensor Selection	The process of selecting sensors that can contribute information to a particular service.

Virtual Sensor	All the physical or virtual items (i.e., services, persons, sensors, GSN nodes), which provide their information through a GSN endpoint.
Sensor Discovery	The process of finding physical and virtual sensors, as well as of the services offered by them.
Resource Discovery	The process of finding an IoT resource (such as a sensor, a service or a database).
Utility Manager	A software component (part of the OpenIoT platform), which performs metering based on the tracking and combination of utility metrics.
Sensor Directory	A software service, which stores, organizes and provides access to information about (physical and virtual) sensors.
(OpenIoT) Sensor Middleware	The part of the OpenIoT middleware platforms that facilitate access to, collection and filtering of OpenIoT data streams.
Global Sensor Networks (GSN)	An open-source sensor middleware platform enabling the development and deployment of sensor services with almost zero-programming effort.
Data Streams	A stream of digital information stemming from a physical or virtual sensor.
Data Stream Engine	A software component enabling the processing of data streams, as well as the management of the process of publishing and subscribing to data stream.
Linked Sensor Data	A set of (Semantic Web) technologies for exposing, sharing, and connecting sensor data, information, and knowledge.

1 INTRODUCTION

1.1 Scope

OpenIoT has introduced and developed a novel architecture for IoT/cloud integration, which can be used as a blueprint for the implementation and deployment of IoT services. The blueprint implementation of the OpenIoT architecture provides the means for integrating virtually any sensor and data stream to the Cloud, while also enabling the dynamic discovery of sensors and their data, along with their composition into IoT services. For this, OpenIoT provides APIs for the development and deployment of IoT services based on sensor data and metadata that reside on the cloud-based (W3C SSN compliant) ontologies of the project. One of the main objectives of the project is to provide a range of tools for the development, deployment and management of OpenIoT-compliant applications and services, i.e., application/services adopting the OpenIoT architecture. Furthermore, the project has promised to integrate these tools in a unified development environment, which would facilitate application developers, solution providers and integrators of IoT services to leverage the capabilities of the OpenIoT middleware platform through minimal programming effort. The purpose of the present document is to report on the OpenIoT Integrated Development Environment (IDE), which comprises the above-mentioned tools. The OpenIoT IDE is an integral component of the OpenIoT open source project, which is currently available at GitHub: http://www.github.com/OpenIotOrg/openiot. The document is accompanying and documenting the prototype implementation of the OpenIoT IDE as part of Deliverable D4.4 of the project. Note that the prototype implementation of the IDE is already available as part of the OpenIoT open-source project outlined above.

While the OpenIoT middleware platform provides the basis for the deployment and operation of IoT services based on Semantic Web technologies, the OpenIoT IDE complements the middleware platform with a range of tools that enable a visual development of IoT applications and services. In particular, the OpenIoT IDE enables the development of simple applications without programming. At the same time, it accelerates the development of more complex applications through minimizing the programming/effort required. To this end, the OpenIoT IDE is in several cases characterized as a “zero-programming” environment. The OpenIoT IDE comprises several development and deployment tools, which are described in detail in later sections.

1.2 Audience

This document addresses the following audiences:

- Users of the OpenIoT platform and tools, including users of the open-source community. For this audience, the document can provide valuable insights into the operation and the merits of the various tools that compose the OpenIoT IDE, as well as on the added value of their integration in a uniform environment.
- Contributors to the OpenIoT platform and tools, including open-source contributors. For this audience, the document provides all the essential information they need to get started with the OpenIoT IDE and tools.
- IoT researchers, given that the OpenIoT IDE is one among very few "zero-programming" frameworks for IoT applications, and probably the only one providing essential support for semantic technologies.
- Researchers participating in IoT projects (e.g., projects of the IERC cluster), which could find the functionalities of the IDE very useful. We expect that the OpenIoT IDE will be an appealing feature of the OpenIoT project, which could attract users and contributors to the open-source project.

1.3 Summary

The OpenIoT IDE operates in line with the OpenIoT architecture and over the middleware modules of the OpenIoT middleware platform. In particular, the various tools of the IDE manage entities, configure parameters, and create applications over OpenIoT-compliant systems. The description of the OpenIoT IDE as part of this document refers to functionalities and services implemented as part of the first release of this deliverable (D4.4.1). A range of improvements to the various tools will be carried out in the coming months and will be included in the second and final version of the present deliverable (D4.4.2).

The present document starts with a presentation of the host environment of the OpenIoT IDE. This host environment provides the means for the integration of the various tools, which leads to multiplicative benefits to the productivity of the developer/integrator, comparing to the use of individual tools alone. The description of the host environment includes information about the Web-based functionalities of the IDE, including information on the functional design and layout of the Web components comprising the IDE. The host environment is carefully designed according to principles of popular IDEs.

Following the presentation of the host environment of the IDE, the present document provides detailed information for each one of the individual modules and tools that compose the first release of the IDE. These include:

(A) The Request Definition module, which allows developers/integrators to formulate IoT services in a visual fashion (i.e., without programming effort),

(B) The Request Presentation module, which allows end users to visualize their services on the basis of libraries of popular/available mashup components,

(C) The Sensor Schema Editor, which facilitates developers and integrators to define and semantically annotate sensors/ICOs that are deployed within the OpenIoT platform,

(D) Tools enabling the configuration of virtual sensors at the lowest levels of the OpenIoT architecture, i.e., where sensors/ICOs and their data stream are integrated into the X-GSN middleware,

(E) Modules enabling the monitoring of the operations of the OpenIoT platform. The presentation of the various modules includes information for downloading and using them from the repository of the open-source project, along with information for deploying them and executing them over the OpenIoT middleware platform.

As part of the second and final release of the present deliverable, the OpenIoT IDE will be enhanced with additional functionalities and tools. At the same time, we expect the already released functionalities to be fine-tuned based on feedback from the open-source community, but also from the process of validating OpenIoT in the scope of real-life use case in WP6 of the project.

1.4 Structure

The deliverable is structured as follows:

- Section 2, following this introduction, positions the OpenIoT IDE (and its individual tools) within the overall OpenIoT architecture. It therefore illustrates the role of the various tools in the OpenIoT architecture, as well as their interfaces to the middleware components of the OpenIoT open-source project.
- Section 3 is devoted to the detailed presentation of the OpenIoT IDE, in terms of the core environment that hosts and integrated the various tools, but also in terms of the individual tools that have been developed and integrated so far. For each tool, the section includes analytical instructions regarding its deployment and use. Technical details are also provided at a reasonable depth.
- Section 4 is the concluding section of this deliverable. Apart from summarizing the main conclusions, the section provides an outlook regarding future developments and enhancements to the IDE, which will lead to a second release during the project (and to the enhanced and final version of this deliverable).

2 ROLE WITHIN OPENIOT ARCHITECTURE

2.1 Overview of OpenIoT Architecture

The OpenIoT architecture is comprised by seven main elements that belong to three different logical planes, as illustrated in **Figure 1** below. These planes are the Utility/Application Plane, the Virtualized Plane and the Physical Plane which include the following modules:

Utility/Application Plane

- The **Request Definition** component enables on-the-fly specification of service requests to the OpenIoT platform by providing a Web 2.0 interface. It comprises a set of services for specifying and formulating such requests, while also submitting them to the Global Scheduler.
- The **Request Presentation** component selects mashups from an appropriate library in order to facilitate a service presentation in a Web 2.0 interface. In order to visualize these services it communicates directly with the Service Delivery & Utility Manager so as to retrieve the relevant data.
- The **Configuration and Monitoring** component enables the management and configuration of functionalities over the sensors and the (OpenIoT) services that are deployed within the OpenIoT platform. Moreover, it enables the user to monitor the health of the different deployed modules.

Virtualized Plane

- The **Scheduler** processes all the requests for services from the Request Definition and ensures their proper access to the resources (e.g., data streams) that they require. This component undertakes the following tasks: it discovers the sensors and the associated data streams that can contribute to service setup; it manages a service and selects/enables the resources involved in service provision.
- The **Cloud Data Storage** (Linked Stream Middleware Light, LSM-Light) enables the storage of data streams stemming from the sensor middleware thereby acting as a cloud database. The cloud infrastructure stores also the metadata required for the operation of the OpenIoT platform (functional data). The prototype implementation of the OpenIoT platform uses the LSM Middleware, which has been re-designed with push-pull data functionalities and cloud interfaces for enabling additional cloud-based streaming processing.
- The **Service Delivery & Utility Manager** performs a dual role. On the one hand, it combines the data streams as indicated by service workflows within the OpenIoT system in order to deliver the requested service (with the help of the SPARQL query provided by the Scheduler) either to the Request presentation or a third-party application. To this end, this component makes use of the service description and resources identified and reserved by the Scheduler component. On the other hand, this component acts as a service metering facility, which keeps track of utility metrics for each individual service. This metering functionality will be accordingly used to drive functionalities such as accounting, billing, and

utility-driven resource optimization. Such functionalities are essential in the scope of a utility (pay-as-you-go) computing paradigm, such as the one promoted by OpenIoT.

Physical Plane

- The **Sensor Middleware** (Extended Global Sensor Network**,** X-GSN) collects, filters, combines, and semantically annotates data streams from virtual sensors or physical devices. It acts as a hub between the OpenIoT platform and the physical world. The Sensor Middleware is deployed on the basis of one or more distributed instances (nodes), which may belong to different administrative entities. The prototype implementation of the OpenIoT platform uses the GSN sensor middleware that has been extended and called X-GSN (Extended GSN).

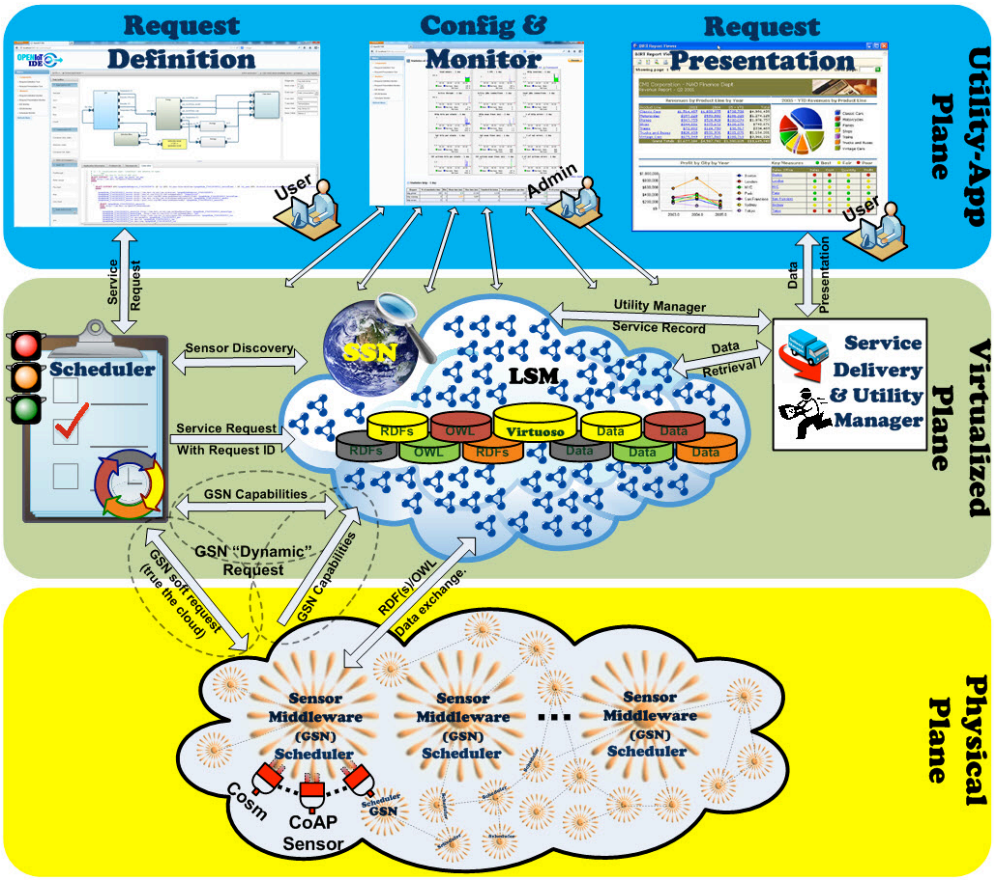

Figure 1 OpenIoT architecture categorized in layers.

These components have been introduced in previous project documents (i.e., D2.2[1], D2.3[2]) and described in more detail in terms of service architecture functional blocks in D4.1[3] and D4.3.1[4]. In this document a more detailed description of the Utility/Application Plane will be provided.

[1] D2.2 Service Delivery Environment Formulation Strategies

[2] D2.3 OpenIoT Platform Requirements and Technical Specifications

[3] D4.1 OpenIoT Detailed Architecture and Proof-of-Concept Specifications

[4] D4.3.1 Core OpenIoT Middleware Platform

2.2 User/Platform Interaction

The OpenIoT architecture was designed with user-friendliness in mind. This mainly involves hiding the complexity to automate and create an environment as pleasant as possible for all the OpenIoT user interfaces and applications. So the core environment of user interaction with the rest of OpenIoT platform is the OpenIoT IDE (**Figure 2** below). This environment resides in the higher tier of the OpenIoT architecture, as seen in Figure 1 above, and is responsible for the sensors, modules management and enabling the configuration of the OpenIoT service delivery capabilities. It provides a single point of entry for all the OpenIoT tools and applications by utilizing Web 2.0 technologies. Currently, the OpenIoT IDE provides the Core Environment, the Request Definition, the Request Presentation, the RDF Schema Editor, and the monitoring tools that are analysed in detail in Chapter 3 below.

Figure 2 IDE Layout and Main Screenshots

3 OPENIOT INTEGRATED DEVELOPMENT ENVIRONMENT

3.1 Host Environment (Core)

The host environment is represented by a central Web-based IDE, which provides common accessibility and functionality for the modules of the entire framework of OpenIoT.

3.1.1 Main Released Functionalities & Services

The layout of the core IDE is that of a classic Web application with a header and dynamic navigation bar with a dynamic content pane. The IDE layout can be seen in Figure 2 above.

3.1.1.1 Web Application Functionalities and Services

The IDE supports the following functionality and services:

- **Component Navigation**

  The IDE provides navigation links from its navigation bar giving access to the deployed OpenIoT components. Selecting a link loads a component dynamically with AJAX in the content pane, through which the user can access its functionality.

- **Loading – Unloading Components**

  A component may be loaded or unloaded currently through the backend. The “refresh menu” button requests the servers for updates to the components and directly changes the available options in the navigation bar.

- **Monitoring**

  In order to implement monitoring features for the various components, the JavaMelody library has been integrated into the system. JavaMelody has the functionality of monitoring the JVM and the Java EE application server. It is a tool to measure and calculate statistics on the real operation of an application depending on the usage of the application by users [Java Melody Project (2013)]. JavaMelody is mainly based on statistics of requests and on evolution charts. Further details on the monitoring component are available in Section 3.5.

3.1.1.2 Web Resources Layout

The Web application is implemented according to the single-page-design principle relying on AJAX to render the parts of the page that change. The various components are loaded in the central content pane according to the source pointed to by the navigation Backing Bean (LayoutController.java). Since the various OpenIoT components have different URL endpoints they are loaded within an iFrame in the central content pane of the page.

The layout of the Web application (which is depicted in **Figure 3** below) is relatively simple since its responsibilities at the moment are simply navigation.

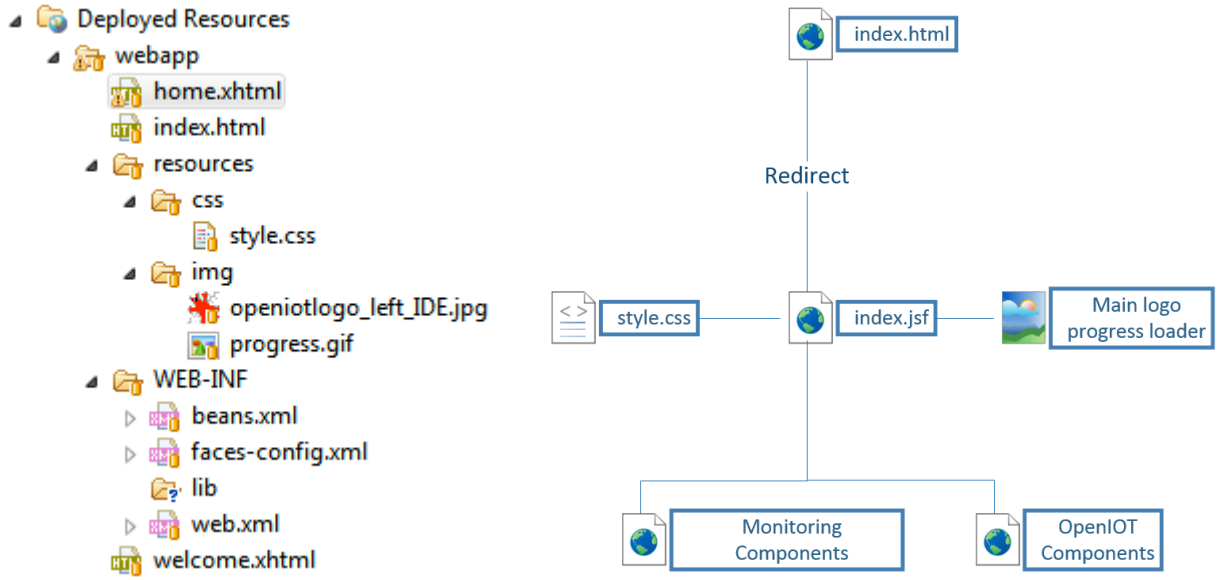

Figure 3 IDE Web layout

As we can see from the diagram, index.html is the first page that is accessed once a request is sent to the application server. Then a redirection goes to index.jsf which provides the application layout and graphics template. Then, depending on the navigation options, OpenIoT or monitoring components are rendered in the central webpage.

3.1.1.3 Web Application UML Class Diagram

The class diagram for the Web application contains a single class which has the responsibility to control navigation and layout (shown in **Figure 4** below).

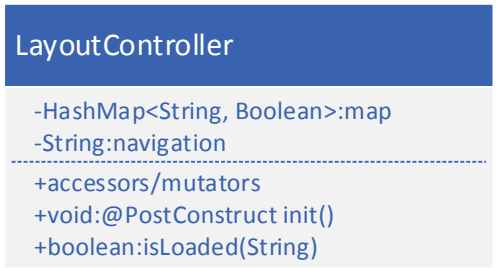

Figure 4 LayoutController controller class

In Table 1 below, we can see the IDE's UML analysis.

Table 1. IDE UML analysis

Service Name	Input	Output	Info
init	none	None	Used to initialize the Session Scoped bean with its starting values.
isLoaded	String	Boolean	Used to scan the HashMap which holds which components are displayed and determines whether to display the link for the particular component
getNavigation	none	String	Retrieves which URL to render in the central content pane
setNavigation	String	None	Sets which URL to render in the central content pane

3.1.2 Download, Deploy & Run

3.1.2.1 Developer

3.1.2.1.1 System Requirements

The requirements for this project is Java 7.0 (Java SDK 1.7) or higher, Maven 3.0 or higher. The application this project produces is designed to be run on JBoss Enterprise Application Platform 6 or JBoss AS 7.1.

3.1.2.1.2 Download

To download the IDE source code, use your favourite git client and pull the code from one of the following links:

- HTTPS: https://github.com/OpenIotOrg/openiot.git
- SSH: git@github.com:OpenIotOrg/openiot.git

The Core IDE module is available under the “openiot/ui/ide/ide.core/” folder

3.1.2.1.3 Deploy From the Source Code

If you have not yet done so, you must configure Maven before testing the IDE Core deployment. After that:

- Start JBoss Enterprise Application Platform 6 or JBoss AS 7.1 with the Web profile
  1. Open a command line and navigate to the root of the JBoss server directory.
  2. Use the following command line to start the server with the Web profile:
     - For Linux: JBOSS_HOME/bin/standalone.sh
     - For Windows: JBOSS_HOME\bin\standalone.bat

- Build and deploy the IDE Core
    - NOTE: The following build command assumes that you have configured your Maven user settings. If you have not, you must include Maven setting arguments in the command line.

    1. Make sure you have started the JBoss Server as described above.
    2. Open a command line terminal and navigate to the root directory of the IDE project containing the pom.xml.
    3. Type this command to build and deploy the archive:
        - mvn clean package jboss-as:deploy
    4. This will deploy “target/ide.core.war” to the running instance of the server.

- Access the application
    - The application will be running at the following URL: http://localhost:8080/ide.core/

- Undeploy the archive
    1. Make sure you have started the JBoss Server as described above.
    2. Open a command line terminal and navigate to the root directory of the “ide.core” project.
    3. When you are finished testing, type this command to undeploy the archive:
        - mvn jboss-as:undeploy

3.1.2.1.4 Run in Eclipse

You can start the JBoss Application Server and deploy the IDE from Eclipse using JBoss Tools or the integrated JBoss Developer Studio. Detailed instructions on how to integrate and start JBoss AS from Eclipse with JBoss Tools are available at the following link:

https://docs.jboss.org/author/display/AS7/Starting+JBoss+AS+from+Eclipse+with+JBoss+Tools

3.1.2.2 User

3.1.2.2.1 System Requirements

The IDE Core supports any modern Web browser (Chrome, Firefox, Safari, Opera, Internet Explorer > IE8).

3.1.2.2.2 Deployment/Undeployment

Deploy: To deploy the IDE, copy the “ide.core.war” to the server's “standalone/deployments” directory.

Undeploy: To undeploy the application, you need to remove the “.deployed” marker file that is generated upon successful deployment of the IDE module

You can find more detailed directions for deployment on JBoss AS7 here: https://docs.jboss.org/author/display/AS7/Application+deployment

3.1.2.2.3 Manual

The use of the IDE is very straightforward. The numbers below in Figure 5 represent different regions/functionalities of the IDE.

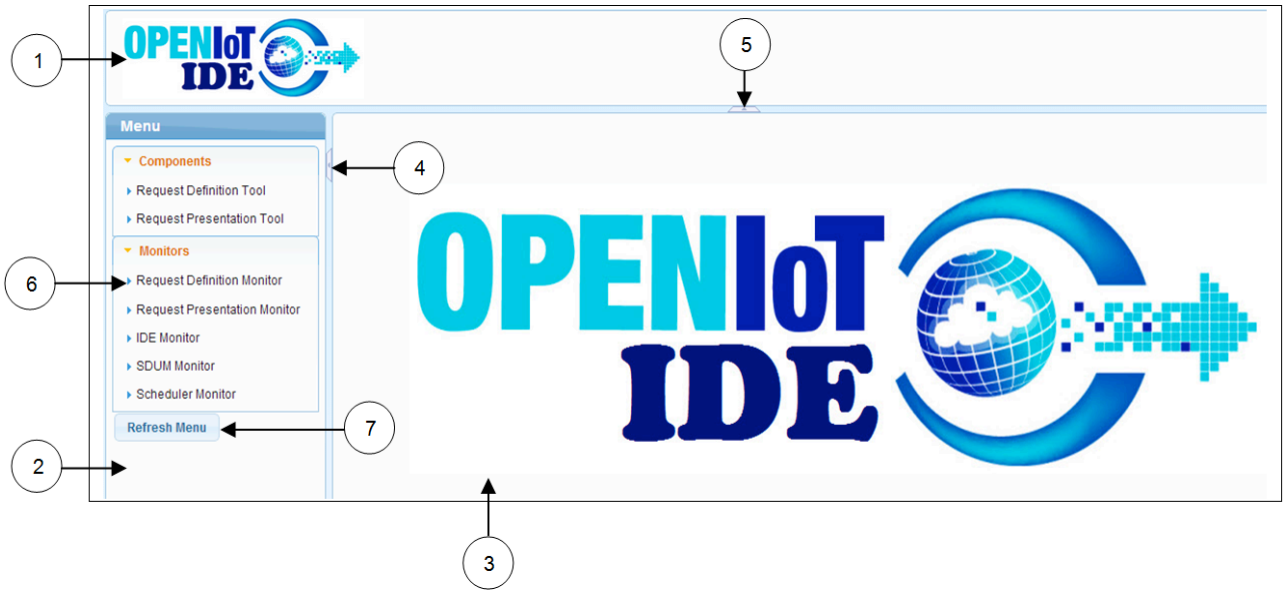

Figure 5 IDE regions

1. The title bar presents the site logo and may possibly host a title or a login component.
2. The navigation pane provides links through the menu panel (6) to the various OpenIoT components or monitors which are rendered in (3).
3. This is the content pane, where the selected components are rendered.
4. This button minimizes the navigation bar.
5. This button minimizes the title bar.
6. The menu panel component displays the various links.
7. The refresh menu button checks if new components have been deployed, or existing ones have been undeployed, and renders the menu panel accordingly.

3.2 Request Definition

The Request Definition module (**Figure 6** below) is a Web application that allows end users to visually model their OpenIoT-based services using a node-based WYSIWYG (What-You-See-Is-What-You-Get) UI (User Interface). Modelled service graphs are grouped into “applications”, i.e., OpenIoT Application Model Objects (OAMOs). These applications are able to group a collection of different services, i.e., OpenIoT Service Model Objects (OSMOs) which comprise/describe a real life application (i.e., weather reports). This enables end users to manage (describe/register/edit/update) different (unrelated) applications from a single point. All modelled services are stored by the OpenIoT Scheduler and are automatically loaded when a user accesses the Web application.

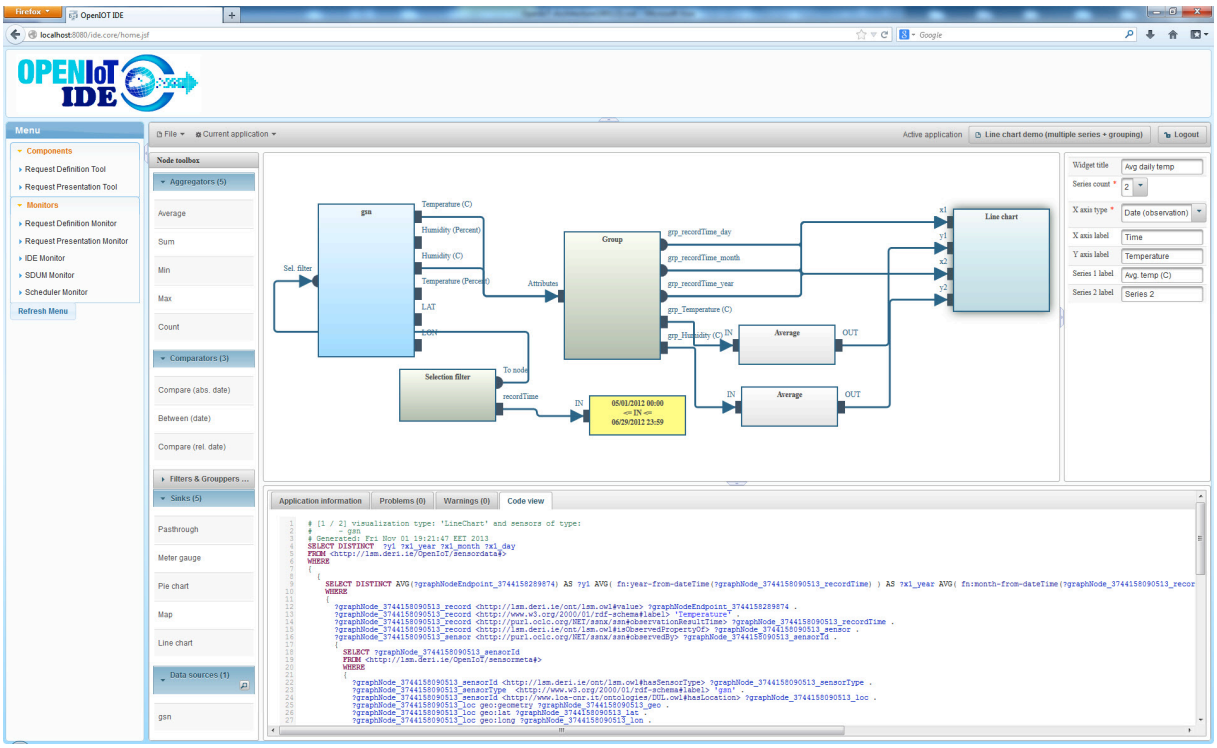

Figure 6 Request Definition view

3.2.1 Main Released Functionalities & Services

Figure 7 below illustrates the main application interface components which are:

- **Menu bar**: provides commands for creating new applications or for opening existing applications for editing. Once an application has been opened for editing, its name will appear on the top right of the menu bar.
- **Central pane**: serves as the workspace area for modelling services.
- **Node toolbox** (left pane): contains the list of nodes that can be dragged into the workspace. Nodes are grouped by their functionality.
- **Properties pane** (right pane): provides access to any selected node's properties.

- **Console pane** (bottom pane): provides workspace validation information (problems/warnings) as well as a debug preview of the generated SPARQL code for the designed service.

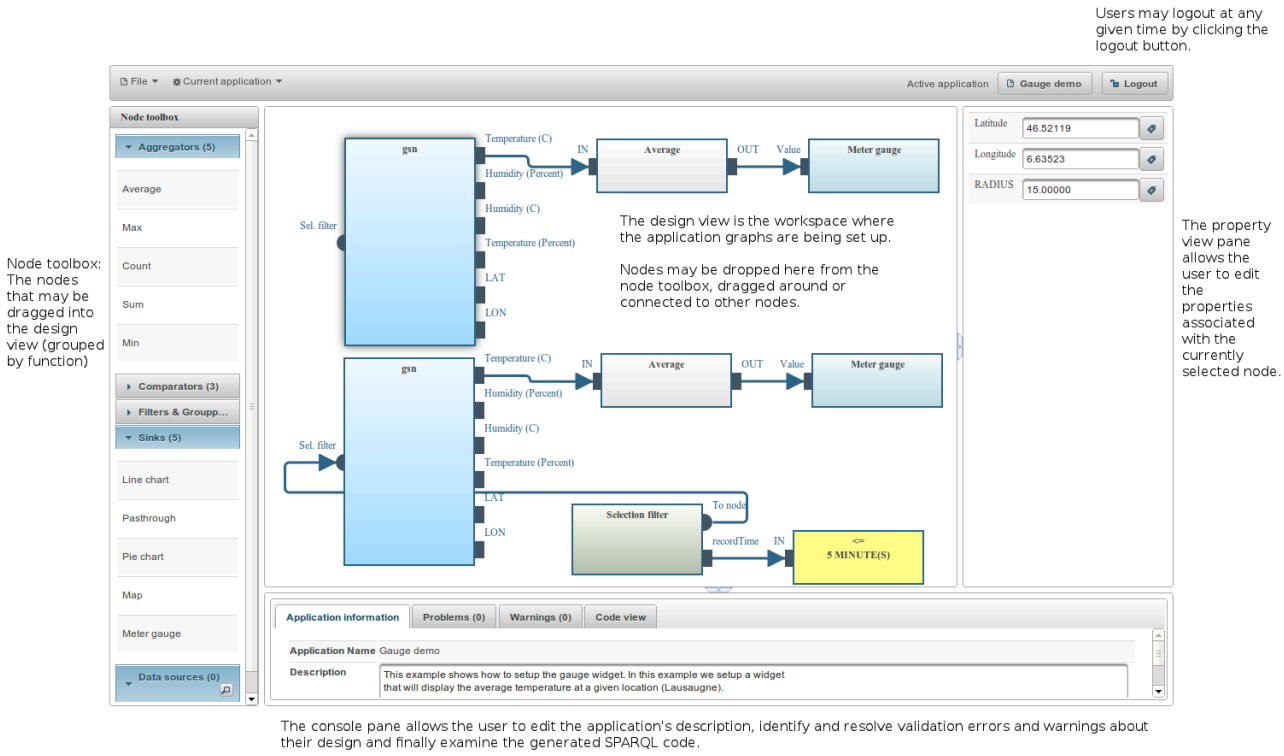

Figure 7 Request Definition main interface components.

3.2.2 Download, Deploy & Run

3.2.2.1 Developer

3.2.2.1.1 System Requirements

All you need to build this project is Java 7.0 (Java SDK 1.7) or higher and Maven 3.0[5] or higher. The application that OpenIoT project produces is designed to be run on JBoss Enterprise Application Platform 6 or JBoss AS 7.1.

3.2.2.1.2 Download

To download Request Definitionmodule's source code, use your favourite git client and retrieve the code from one of the following URLs:

- HTTPS: https://github.com/OpenIotOrg/openiot.git
- SSH: git@github.com:OpenIotOrg/openiot.git

The Request Definition module is available under the "openiot/ui/ui.requestDefinition/" folder and the Request Commons files under "openiot/ui/ui.requestCommons/".

[5] http://maven.apache.org/download.cgi#Installation

3.2.2.1.3 Deploy From the Source Code

Start JBoss Enterprise Application Platform 6 or JBoss AS 7.1 with the Web profile:

- Open a command line terminal and navigate to the root of the JBoss server directory.
- Use the following command line to start the server with the Web profile:
  - For Linux: JBOSS_HOME/bin/standalone.sh
  - For Windows: JBOSS_HOME\bin\standalone.bat

3.2.2.1.3.1 Download and Install/Deploy Dependencies

NOTE: The following build command assumes that you have configured your Maven user settings. If you have not, you must include Maven setting arguments in the command line.

- Download and deploy the Scheduler[6]
- Download and install “ui.requestCommons”
  - Open a command line terminal and navigate to the root directory of the Request Commons project.
  - Type the following command to build and install:
    - “mvn clean package install”

3.2.2.1.3.2 Build and Deploy the Request Definition Web application

NOTE: The following build command assumes you have configured your Maven user settings. If you have not, you must include Maven setting arguments in the command line.

- Make sure you have started the JBoss Server as described above.
- Open a command line terminal and navigate to the root directory of the request definition Project.
- Type this command to build and deploy the archive:
  - “mvn clean package jboss-as:deploy”

This will deploy “target/ui.requestDefinition.war” to the running instance of the server.

3.2.2.1.3.3 Access the Application

- The application will be running at the following URL: “http://servername:8080/ui.requestDefinition/”

[6] https://github.com/OpenIotOrg/openiot/wiki/Global-Scheduler

3.2.2.1.3.4 Undeploy the Archive

- Make sure you have started the JBoss Server as described above.
- Open a command line terminal and navigate to the root directory of the Request Definition project.
- When you are finished testing, type this command to undeploy the archive:
    - "mvn jboss-as:undeploy"

3.2.2.1.4 Run in Eclipse

3.2.2.1.4.1 Integrating and Starting JBoss server

You can start JBoss Application Server and deploy the Request Definition module from Eclipse using JBoss tools. Detailed instructions on how to integrate and start JBoss AS from Eclipse with JBoss Tools are available at the following link: https://docs.jboss.org/author/display/AS7/Starting+JBoss+AS+from+Eclipse+with+JBoss+Tools

3.2.2.1.4.2 Integrating and Deploying Request Definition Module

To integrate and deploy the Request Definition module in Eclipse one should follow the steps below:

1. Import existing maven project "File>Import>Maven>Existing Maven Projects"
2. Click the "Browse" button and navigate to the Request Definition module's source code directory that has been previously downloaded.
3. Select the ui.requestDefinition and click the "Finish" button.
4. Right click on the "ui.requestDefinition" project and select "Run As>Maven Build…"
5. Insert the following parameters:
    a. Goals: "clean package jboss-as:deploy"
    b. Profiles: "arq-jbossas-remote"
    c. Name: "ui.requestDefinition package-deploy" (or your preferred name)
6. Click the "Run" button (the JBoss Server should be already running). The project will automatically build itself, get deployed and run at the JBoss AS running instance. From now on this configuration should be available at the Eclipse Run Configurations under Maven Build.

To undeploy the Request Definition module from the running instance of the JBoss AS follow the steps below:

1. Right click on the "ui.requestDefinition" project and choose "Run As>Maven Build…"
2. Insert the following parameters:
    a. Goals: "jboss-as:undeploy"
    b. Profiles: "arq-jbossas-remote"
    c. Name: "ui.requestDefinition undeploy" (or your preferred name)

Click the "Run" button (the JBoss Server should be already running). The project will automatically be undeployed from the JBoss AS running instance. From now on this

configuration should be available at the Eclipse Run Configurations under Maven Build.

3.2.2.2 Programming

3.2.2.2.1 Defining New Nodes

With the exception of the sensor nodes which are populated dynamically from a sensor discovery query, all other nodes are specifically annotated Plain Old Java Object s (POJOs) that extend the “DefaultGraphNode” class. All node implementations should be placed under the “org.openiot.ui.request.definition.web.model.nodes.impl” package. The system will automatically scan for the annotated POJOs during deployment and populate the node toolbox. The most important annotations are:

- ***@GraphNodeClass***. This annotation marks a POJO as a node that can be used for a service graph. The annotation expects the following attributes:
  - label: the name of the node (localizable).
  - type: the type (group) of the node. Should be one of SOURCE, AGGREGATOR, COMPARATOR, SINK or FILTER.
- **scanProperties**: if set to true, the annotation scanner will automatically initialize the node's properties and endpoints from the @*NodeProperty* and @*Endpoint* annotations.
- ***@NodeProperties***. This annotation defines a list of @NodeProperty annotations.
- ***@NodeProperty***. This annotation defines a node property. The annotation expects the following attributes:
  - **type**: one of the PropertyType enumerations. Specify if the property is readable, writeable or both.
  - **javaType**: The fully qualified name of the java type that stores this property's value.
  - **name**: the name of the property (localizable).
  - **required**: set to true if the property is required, false otherwise
  - **allowedValues**: an optional attribute that specifies an array of allowed variables to be selected by a drop-down menu. In this case, the Java type attribute should be java.lang.String.
- ***@Endpoints***: This annotation defines a list of @*Endpoint* annotations.
- ***@Endpoint***: This annotation defines a node's endpoint. It expects the following attributes:
  - **type**: one of the EndpointType enumerations. Specify if the endpoint serves as an input or an output.
  - **anchorType**: one of the AnchorType enumerations. Specify the location of the endpoint on the rendered node.

- **scope**: specifies the types of endpoints that can connect to this endpoint (if this is an input) or the types of endpoints to which this endpoint can connect (if this is an output).
- **maxConnections**: the maximum number of connections that can originate from this node (if this is an output) or end to this node (if this is an input).
- **label**: the label of the endpoint (localizable).
- **required**: set to true if this endpoint requires a connection, false otherwise.

3.2.2.2.2 Localization Support

The Request Definition application has full i18n localization support via property files. These files are placed under the org.openiot.ui.request.definition.web.i18n package. The following rules are used for localizing node elements:

- All node localization entry labels are defined as the concatenation of the prefix "*UI_NODE_*" and the endpoint label's name as defined in the POJO annotations. For example, an endpoint with label TEST has the localization label "*UI_NODE_ENDPOINT_TEST*".
- All node localization entry labels are defined as the concatenation of the prefix "*UI_NODE_ENDPOINT_*" and the endpoint label's name as defined in the POJO annotations. For example, an endpoint with label TEST has the localization label "*UI_NODE_ENDPOINT_TEST*".
- All node localization entry labels are defined as the concatenation of the prefix "*UI_NODE_PROPERTY_*" and the property label's name as defined in the POJO annotations. For example, a property with label TEST has the localization label "*UI_NODE_PROPERTY_TEST*".

3.2.2.3 User

3.2.2.3.1 System Requirements

All you need to run this project is Java 7.0 (Java SDK 1.7) or higher and JBoss Enterprise Application Platform 6 or JBoss AS 7.1.
You can download the binaries through the OpenIoT Wiki[7] under the Users>Downloads[8] section.

3.2.2.3.2 Deployment/Undeployment

3.2.2.3.2.1 JBoss AS 6.0

Deploy: To deploy the Request Definition module on JBoss AS 6.0, copy the "ui.requestDefinition.war" to the server's "deploy" directory.

[7] https://github.com/OpenIotOrg/openiot/wiki

[8] https://github.com/OpenIotOrg/openiot/wiki/Downloads

Undeploy: Remove the application war file (ui.requestDefinition.war) from the JBoss deploy directory while the server is running.

3.2.2.3.2.2 JBoss AS 7.0

Deploy: To deploy the Request Definition module on JBoss AS 7.0, copy the "ui.requestDefinition.war" to the server's "standalone/deployments" directory.

Undeploy: To undeploy the application, you need to remove the ".deployed" marker file that is generated upon successful deployment of the Request Definition module.

You can find more detailed directions on the ins and outs of deployment on JBoss AS7 here: https://docs.jboss.org/author/display/AS7/Application+deployment

3.2.2.3.3 Manual

3.2.2.3.3.1 Login or Register

In order to use the Request Definition UI, a valid OpenIoT account is required. When the user first launches the Request Definition UI, they will be automatically redirected to a login screen (**Figure 8**). At this point, users may either login using their email and password credentials or register for a new account. After a successful login or registration, the user will be redirected to the mail application UI.

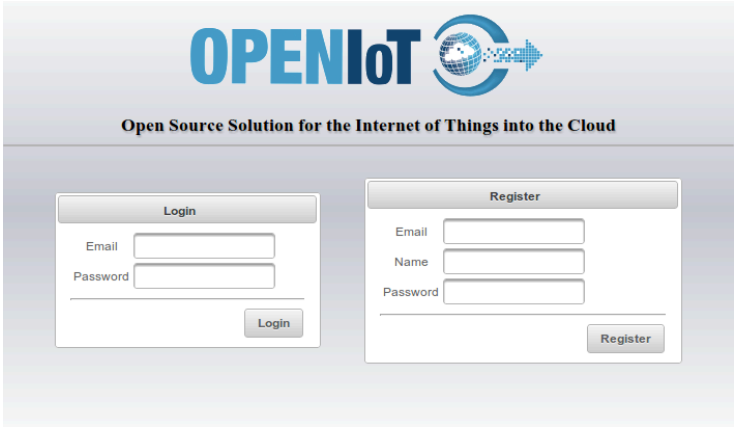

Figure 8 Request Definition

3.2.2.3.3.2 UI Overview

The main application UI consists of four panes: the node toolbox, the property view pane, the design view workspace and the console pane (**Figure 7** above).

3.2.2.3.3.2.1 The Node Toolbox

The node toolbox is located on the left side of the application UI. It contains a list of nodes that may be dropped into the design view workspace. Nodes are grouped by their function.

The data source group contains a list of sensors types that were discovered via a location-based query. To perform a new discover query, click the magnifying glass icon next to the data source header.

3.2.2.3.3.2.2 The Property View Pane

The property view pane is located on the right side of the screen. It allows users to examine and modify the properties of the currently selected node in the design view workspace.

Depending on the selected node type, the UI may allow the user to convert a property field into a variable (**Figure 9**, **Figure 10**). Variable properties support application cases where one or more of the field values are known during the query execution time and not during the application design time. This functionality is, however, not supported for sink nodes.

To convert a property field into a variable, click the “Convert to variable” button next to the property field to launch the variable editor dialog. Check the “Convert to variable” checkbox, enter a name for the variable and adjust its default value. When you click the “Apply” button, the previous variable field will become read-only and contain the variable name.

To revert this action, click again the “Convert to variable” button next to the property field and uncheck the “Convert to variable” checkbox.

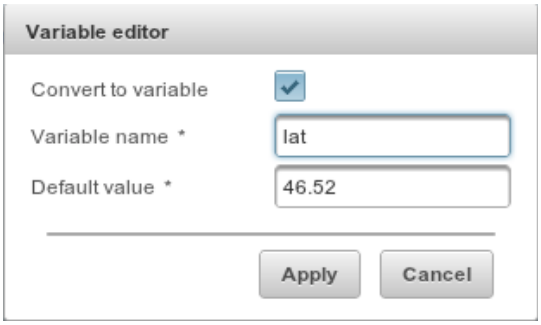

Figure 9 Request Definition property view pane (1)

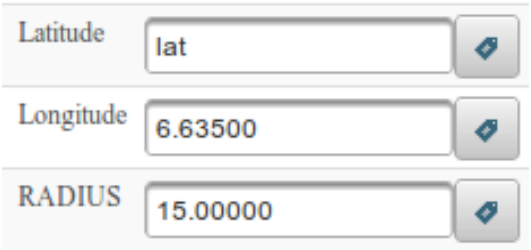

Figure 10 Request Definition property view pane (2)

3.2.2.3.3.2.3 The Design View

The design view workspace serves as a canvas where toolbox nodes can be dropped, re-arranged (click&drag) and connected together.

Connections between nodes may be established by clicking on the source node endpoint and then dragging a connection to a destination endpoint. When a connection attempt is performed, the UI will automatically highlight the destination endpoints that can be connected.

Node selection is performed by clicking on a node. To deselect the currently selected node, click anywhere inside the canvas area. To delete a node, select it, and press the “Delete” key on your keyboard. To disconnect a connection, click the connection destination endpoint, drag the connection, and drop it to the canvas.

3.2.2.3.3.2.4 The Console Pane

The console pane allows users to read and edit the application's description, identify and resolve validation error and warnings, and examine the generated SPARQL code.

3.2.2.3.3.3 The application Menus

3.2.2.3.3.3.1 File Menu

In **Figure 11**, we can see the File menu.

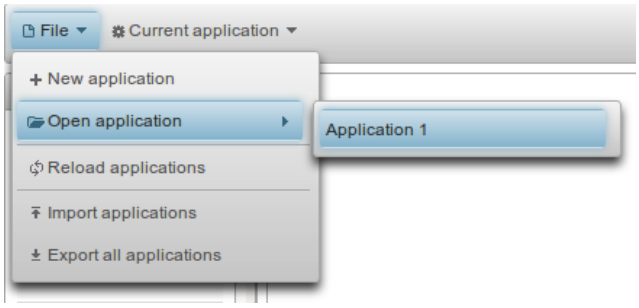

Figure 11 Request Definition File menu

New application

To create a new application, click the “New application” menu option to launch the “New application” dialog (**Figure 12**).

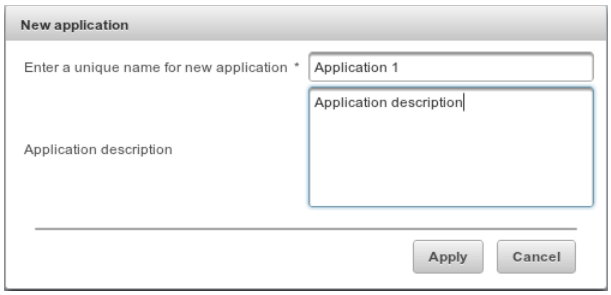

Figure 12 Request Definition “New application” dialog

Enter a name for the application and an optional description. The name should be unique. Click the “Apply” button to create the new application and to open it into the workspace view.

Open application

To open a specific application into the workspace click the “Open application” menu and select one of the available applications. The selected application will be loaded into the workspace view.

Reload applications

This menu option allows the user to reload their applications from the OpenIoT repositories. The loaded applications will replace the currently loaded applications.

Import applications

This menu options allows the user to import a set of applications that were exported by another Request Definition instance as a XML document. By clicking this option, the “Import applications” dialog (**Figure 13**) will appear. Choose a file to upload and select whether the applications should be just imported into the workspace or also be persisted to the OpenIoT repositories. After clicking “Apply”, the application XML file will be loaded and replace the currently loaded applications.

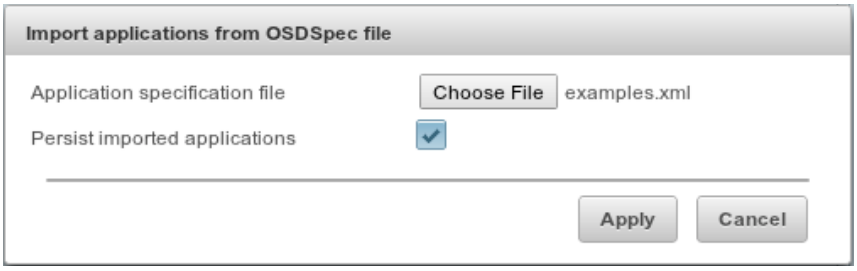

Figure 13 Request Definition “Import application”dialog

Export applications

This menu option allows the user to export the currently loaded set of applications as an XML document which can then be imported by another Request Definition UI instance. Before the applications are exported, the system will first validate them. If any validation error occurs, the system will display an error message and abort the export process.

3.2.2.3.3.3.2 The Current Application Menu

In **Figure 14**, we can see the “Current application” menu.

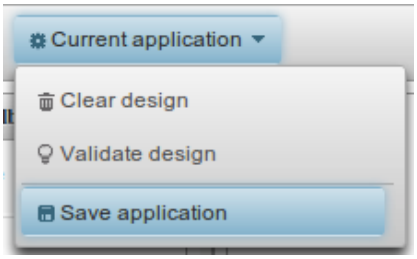

Figure 14 Request Definition “Current application” menu

Clear design

This menu option allows the user to clear the currently opened application. Any unsaved changes will be lost.

Validate design

This menu option will validate the currently opened application and display a validation summary message indicating any validation warnings and errors (**Figure 15**). Warnings and errors are displayed inside the “Problems” and “Warnings” tabs in the console pane. By clicking on a problem or warning description, the UI will highlight the problematic node or endpoint.

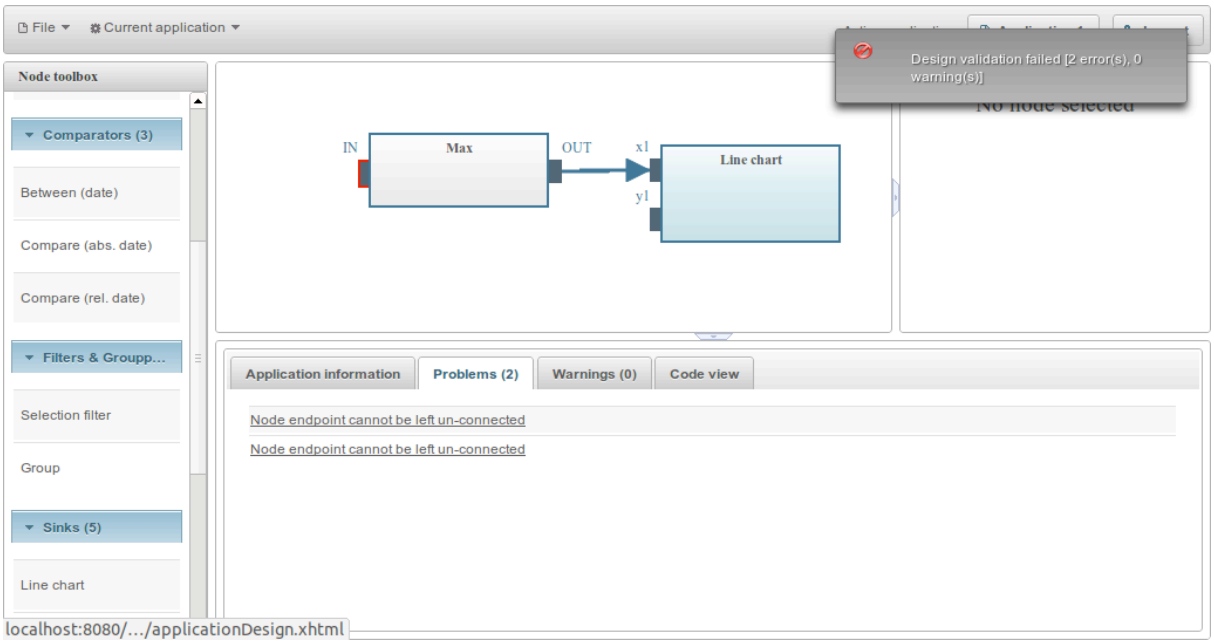

Figure 15 Request Definition design validation

If the validation is successful, the “Code view” tab in the console pane (**Figure 16**) will display the generated SPARQL code.

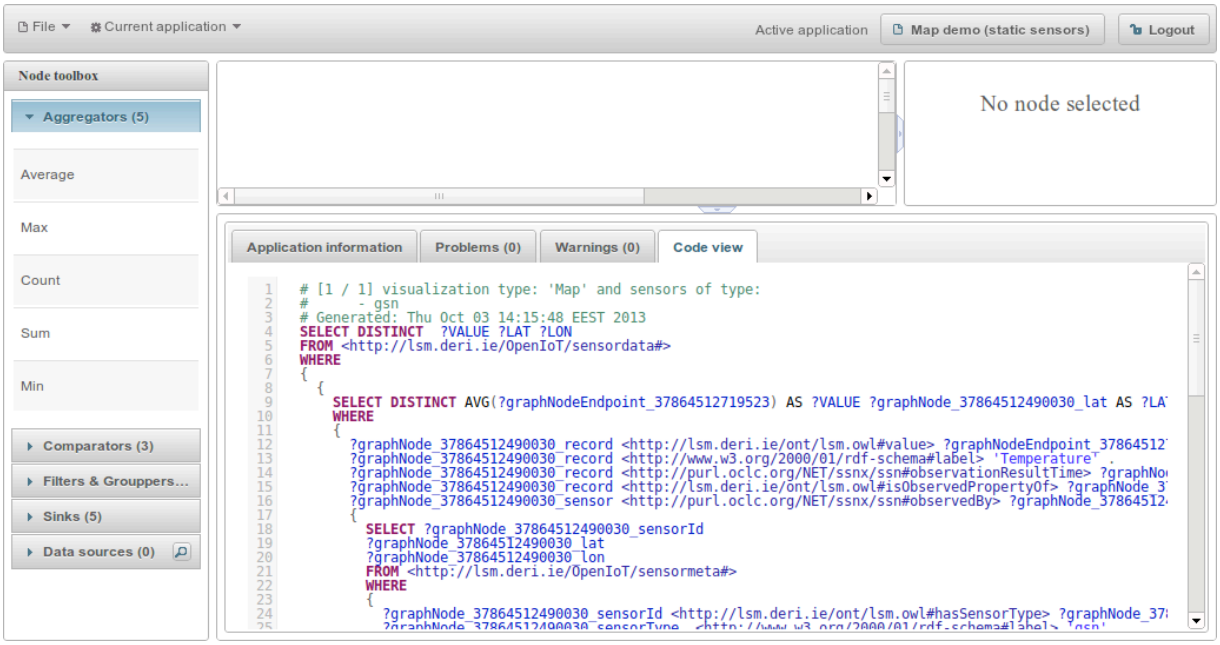

Figure 16 Request Definition validation and “Code view” tab

Save application

This menu option allows the user to save the currently edited application to the OpenIoT repositories so that it can be accessed by the Request Presentation UI and other third-party applications. Before the application is sent to the OpenIoT scheduler registration service, it will first be validated. If any validation errors are detected, the system will prompt the user to correct them and then try saving the application again.

3.2.2.3.3.4 Sensor Discovery

As you may notice, the data sources entry in the node toolbox initially contains no entries. Since the OpenIoT architecture supports a vast number of distributed sensor types (classes) and showing them all at the same time is not feasible, the user needs to select a subset of the available sensors using a location-based search query. To perform a search, click on the magnifying glass icon next to the data sources header in the node toolbox. This will launch the “Discover sensor” dialog (**Figure 17**).

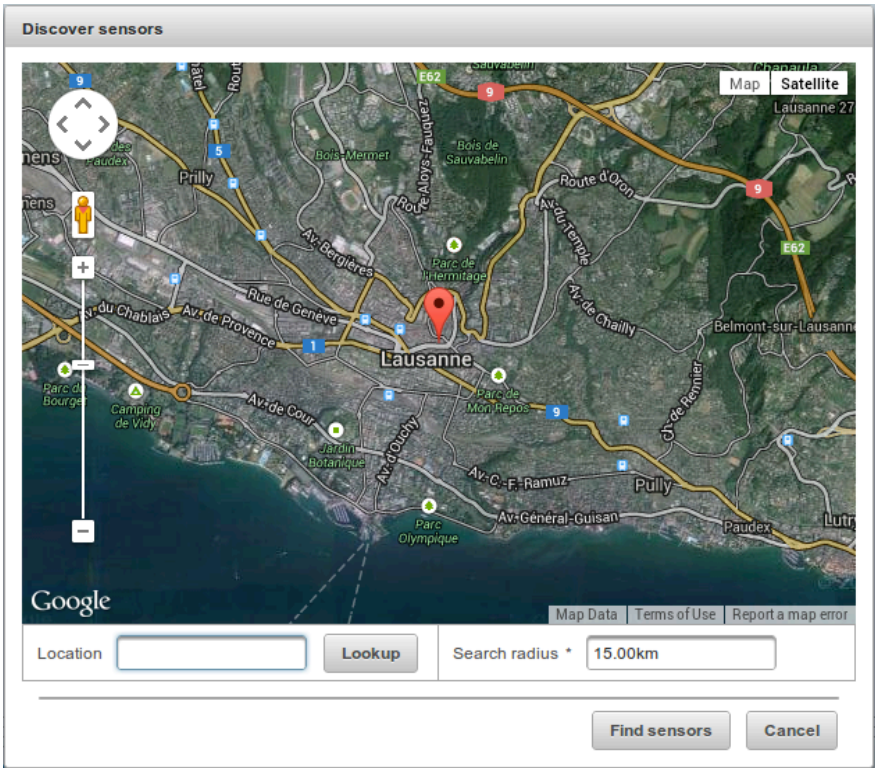

Figure 17 Request Definition ”Discover sensor” dialog

Sensor discovery is facilitated by selecting a geographical location and specifying a search radius. To select the search centre location, type the location's name into the location text field and then click the lookup button. The map will then display the location that matched your search query. Click on any map point to place a selection marker. Then specify a value for the search radius field and click the “Find sensors” button. The system will execute the sensor discovery query and populate the data sources group in the node toolbox with any sensor type (class) that matched your search query. These sensor nodes can then be dragged to the main workspace view.

3.2.2.3.4 Requirements for Designing OpenIoT Applications Using the UI

Any application design should contain at least one source node and one sink node. Source nodes can be discovered using the “Discover sensors” dialog. Sink nodes are basically visualization widgets that will retrieve the incoming (processed) data from the connected source nodes and display them inside a dynamically generated dashboard (Request Presentation UI)

3.2.2.3.4.1 Available Sink Nodes

3.2.2.3.4.1.1 Meter Gauge Node

The meter gauge node displays its input using a meter gauge. Its “Value” input endpoint can only accept aggregated values (an aggregation node is required between the source node and this node). The meter gauge supports the following editable properties (shown in **Figure 18**):

- Widget title: The title that will appear in the Request Definition dashboard
- Measure unit: A label that will appear next to the gauge to help users understand the measurement unit of the displayed value
- Min/Max value: The minimum and maximum expected value for the incoming data

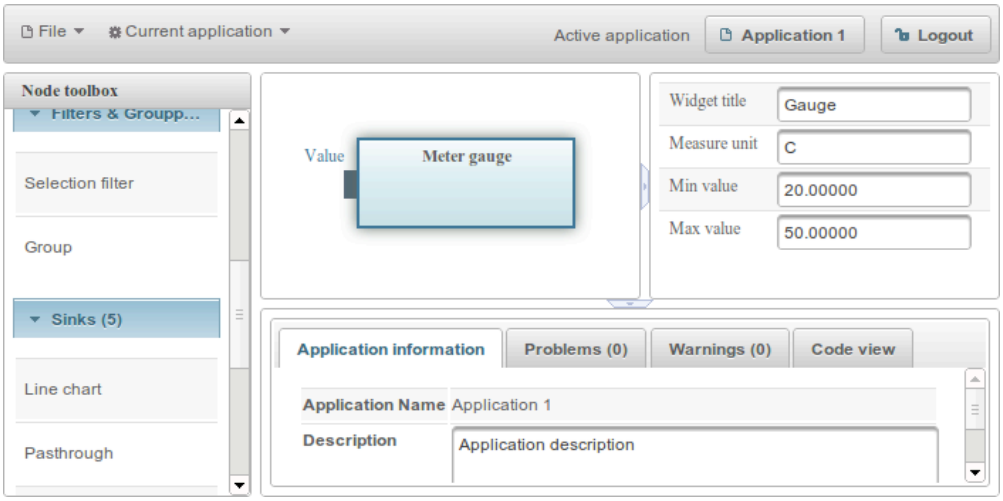

Figure 18 Request Definition gauge node usage

3.2.2.3.4.1.2 Map Node

This node displays a map where the node's inputs are displayed using markers or circle overlays. The node exposes endpoints for connecting the latitude and longitude endpoints of a discovered sensor. Its “Value” input endpoint can only accept aggregated values (an aggregation node is required between the source node and this node's “Value input”). The map node supports the following editable properties (shown in **Figure 19**):

- Widget title: The title that will appear in the Request Definition dashboard
- Latitude/Longitude: The coordinate pair that will serve as the map's centre for the visualization widget
- Zoom level: The map zoom level for the visualization widget
- Type: The type of the overlays that will be displayed on the map; available options are “Markers only”, “Circles only”, and “Circles and markers”
- Value scaler: The rendered circle overlays' radius needs to be specified in pixels. The value scaler allows us to normalize the input values so that the rendered circles appear with the desired size. For example, if the displayed value is greater than 0 and less than 1.0, we can apply a value scaler equal to 100.0 to plot circles between 0 and 100 pixels.

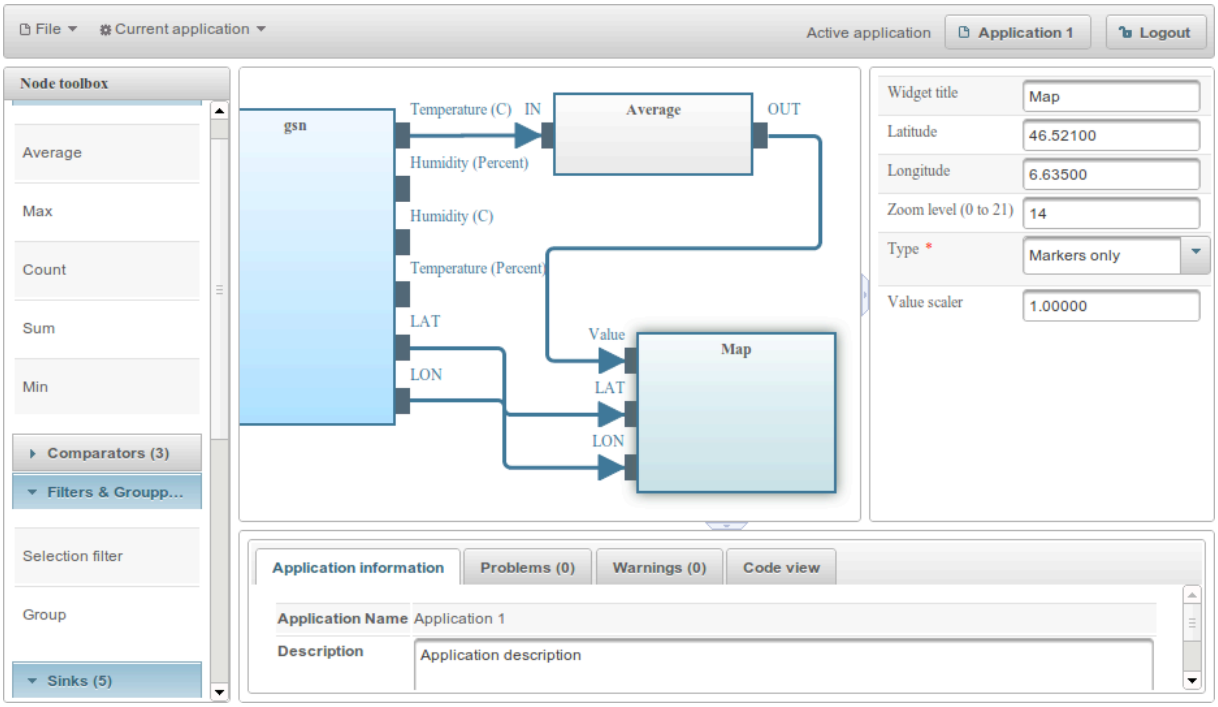

Figure 19 Request Definition map node usage

3.2.2.3.4.1.3 Pie Chart Node

This node displays a pie chart with a configurable number of series. Each series' input endpoint can only accept aggregated values (an aggregation node is required between the source node and this node's series value input). The node supports the following editable properties (shown in **Figure 20**):

- Widget title: The title that will appear in the Request Definition dashboard
- Series count: The number of series. When this value is changed, the system automatically will generate/remove series endpoints depending on the selected value. The system will also generate/remove properties for each series' label.
- Series X label: A label for each one of the configured series

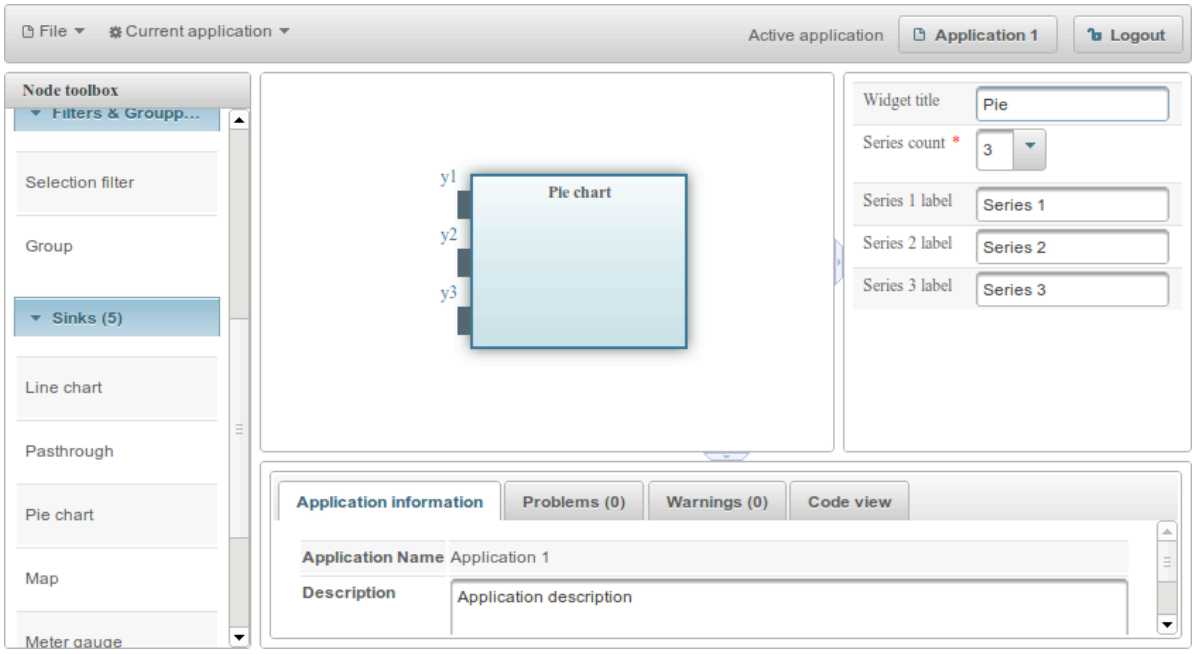

Figure 20 Request Definition pie chart node usage

3.2.2.3.4.1.4 Line Chart Node

This node displays a line chart that supports multiple user-configurable series. Depending on the type of the X-axis input, the line chart operates in 3 modes. This node supports the following properties:

- Widget title: The title that will appear in the Request Definition dashboard
- Series count: The number of series. When this value is changed, the system will automatically generate/remove series endpoints depending on the selected value. The system will also generate/remove properties for each series' label.
- X-axis type: The type of X-axis. The allowed options are “Number”, “Date (result set)” and “Date(observation)”.
- Series X label: A label for each of the configured series.

3.2.2.3.4.1.5 Line Chart With a Numeric X-axis

In this mode, the X-axis type is set to “Number”. When in this mode, the node will generate a (x_i, y_i) endpoint tuple for each configured series. All endpoints expect an aggregated numeric value to be connected to them. This mode of operation allows us to setup line charts that plot one attributes value with respect to another attribute's value (**Figure 21**).

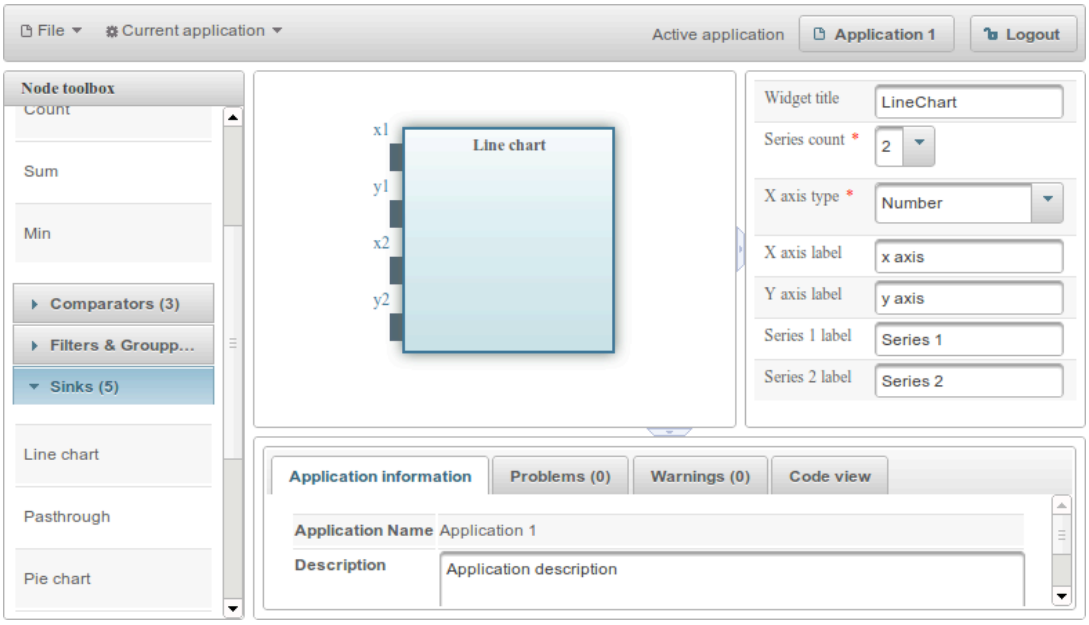

Figure 21 Request Definition line chart usage (1)

3.2.2.3.4.1.6 Line Chart Where the X-axis value is the Timestamp of the Query Response

This mode is enabled by setting the X- axis type to “Date (result set)”. In this mode, the node will only generate a y_i endpoint for each configured series. The visualization widget will use the incoming query response timestamp as the X-axis value (**Figure 22**). This mode of operation allows us to setup line charts that plot an attribute's aggregated value over time (like, for example, the average temperature for the last 5 minutes)

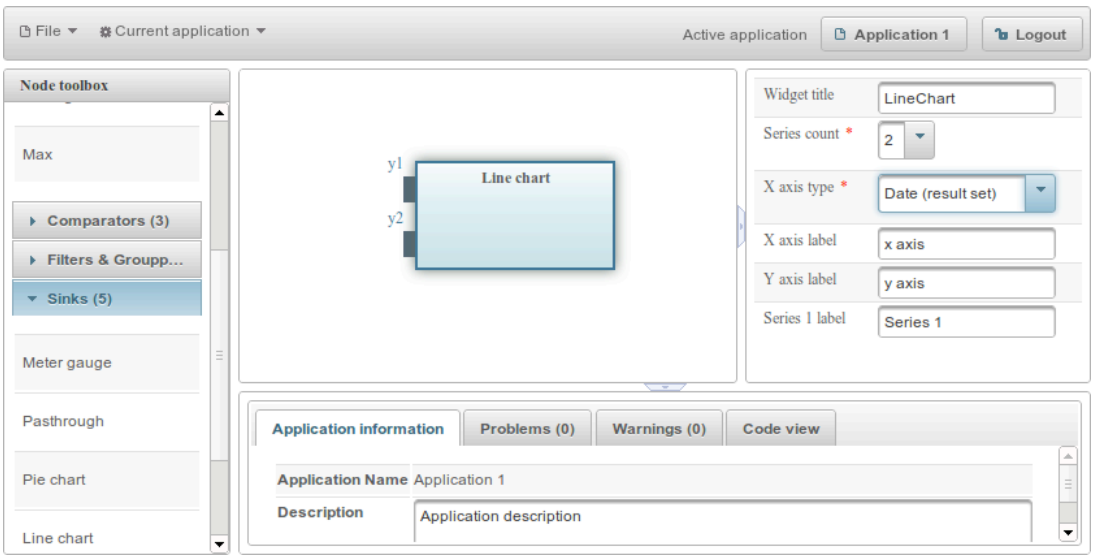

Figure 22 Request Definition line chart usage (2)

3.2.2.3.4.1.7 Line Chart Where the X-axis Value is Generated by a Group Node

This mode is enabled by setting the X-axis type to “Date (observation)”. When in this mode, the node will generate a (x_i, y_i) endpoint tuple for each configured series. The main difference from the other two modes of operation is that in this mode, we can connect multiple time groups (generated by a grouping node) to each one of the x_i endpoints. The time group connection order is not important. The visualization widget will automatically detect the connected time groups from the query results and generate the appropriate timestamp for the X-axis. The y_i endpoints expect an aggregated numeric value generated by a grouping node to be connected to them (**Figure 23**).

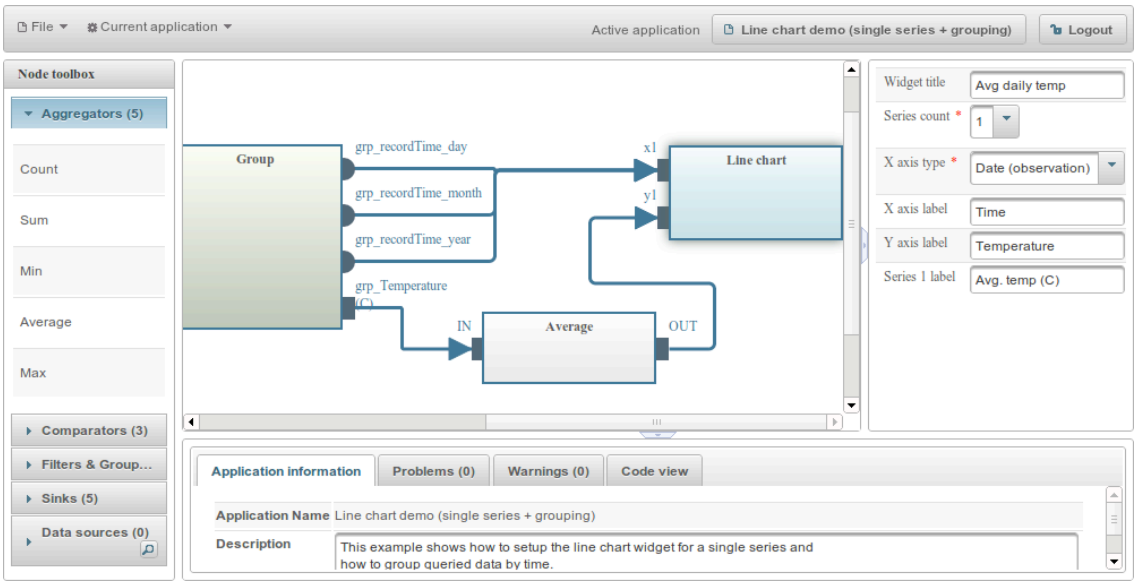

Figure 23 Request Definition line chart usage (3)

3.2.2.3.4.1.8 Passthrough Node

This is a special node that works as a generic sink. It allows the user to define a configurable number of attribute endpoints and connect values to each one of the attributes. This node was designed to support third-party application developers that want to use the Request Definition UI to visually design their applications and then directly query the middleware for results so as to update their custom UIs. The visualization widget for this particular node renders all results in tabular form. This node supports the following properties (shown in **Figure 24**):

- Widget title: The title that will appear in the Request Definition dashboard
- Attribute count: The number of attribute endpoints. When this value is changed, the system will automatically generate/remove endpoints from the node depending on the selected value.

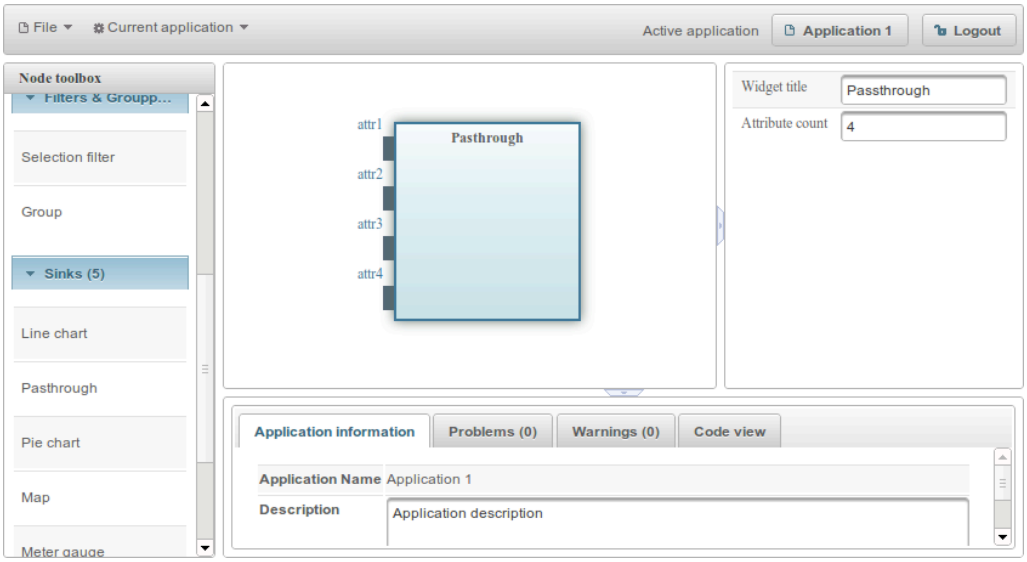

Figure 24 Request Definition passthrough node usage

3.2.2.3.4.2 Aggregator Nodes

The application designer provides several nodes for aggregating values (count, min, max, sum and average). Keep in mind that most sink nodes expect an aggregated value expression as their input. These nodes do not expose any user-editable properties (**Figure 25**).

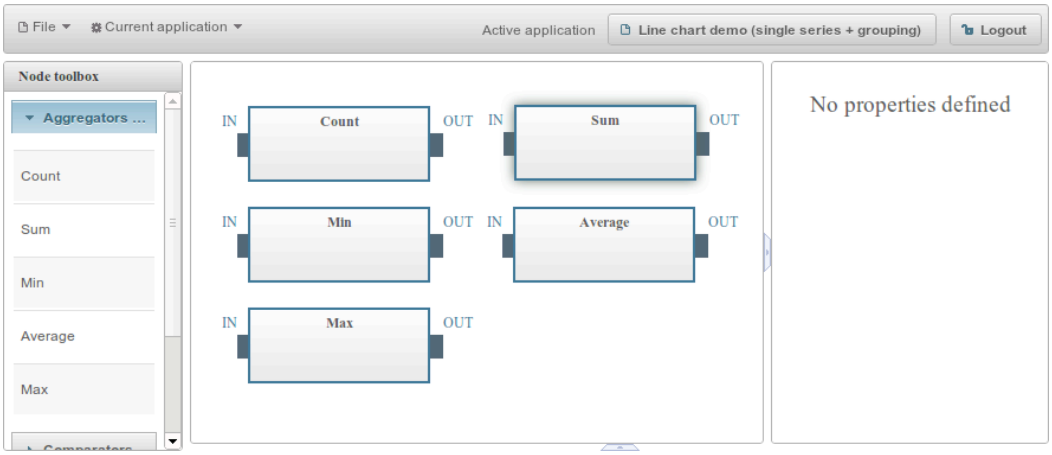

Figure 25 Request Definition aggregator node usage

3.2.2.3.4.3 Filter and Group Nodes

3.2.2.3.4.3.1 Filter Node

The filter node allows us to specify a time-filtering condition for selecting recorded sensor data. To setup data filtering, drag & drop a discovered sensor node to the workspace, then drag & drop a filtering node to the workspace and connect it to node endpoint with the sensor's “Sel. Filter” endpoint. Once the connection is established, the “recordTime” endpoint will appear on the Selection Filter node (**Figure 26**). This

endpoint can be connected to any comparator node to setup the filter conditions (described in the following sections).

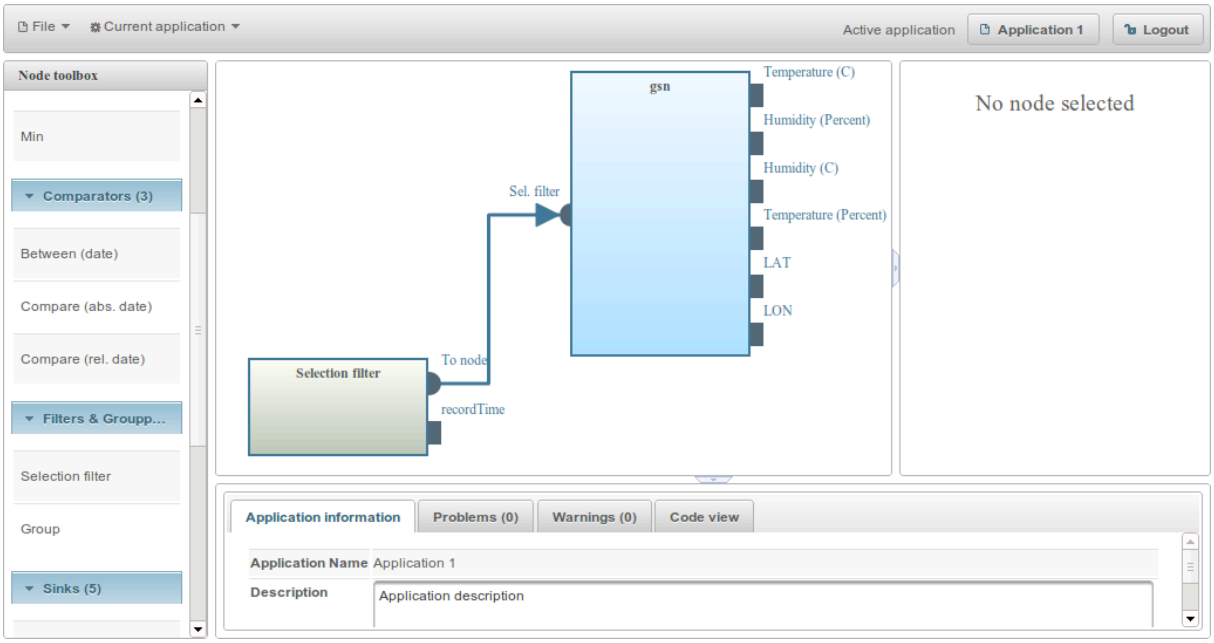

Figure 26 Request Definition filter node usage

3.2.2.3.4.3.2 Group Node

This node allows us to partition one or more sensor attributes into time period buckets. Each grouped value may then be connected to an aggregation node to calculate an aggregate for each generated bucket. To setup this node, drag & drop it into the workspace and then connect one or more sensor attributes to it. Each connected sensor attribute will generate a new output endpoint (with the grp_ prefix). This endpoint represents each aggregated bucket's contents and can be connected in turn to an aggregation node to apply an aggregation function to each bucket.

To select the grouping options click the group node and press the “Grouping options” button in the property pane (**Figure 27**). This will launch the “Grouping options” dialog. The left list contains the available grouping fields while the right list contains the fields that will be used for grouping. Fields can be added and removed from the right list using the provided controls, while the actual group order can be changed by re-arranging the group field order.

Keep in mind that the grouping order is important for getting back proper data (grouping by YEAR, MONTH, DAY is different than grouping by DAY, MONTH, YEAR). Once you have decided which fields should be used for grouping, click the “Apply” button. Depending on the group fields that you selected, one or more time-bucket output endpoints will appear on the group node. These endpoints can be connected to any sink node that accepts time-based inputs (currently only the line chart supports this kind of connections).

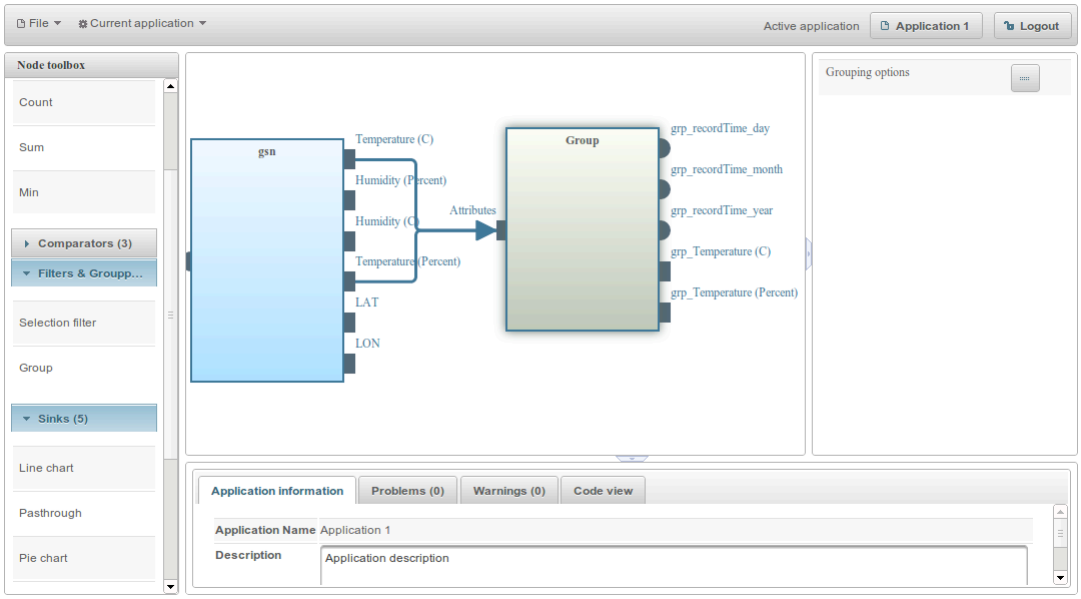

Figure 27 Request Definition group node usage

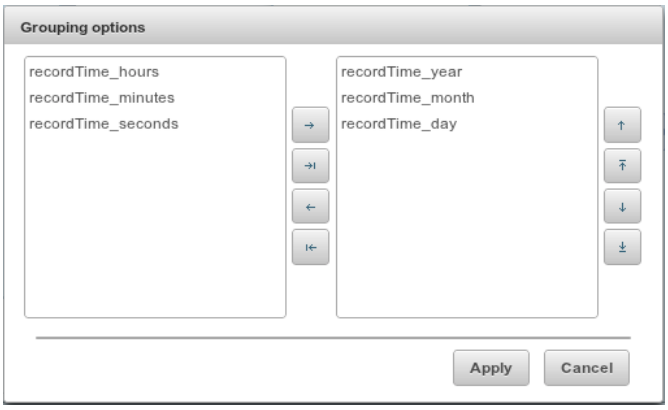

Figure 28 Request Definition “Grouping options” dialog

3.2.2.3.4.4 Selection Filter Nodes

These nodes are used to specify a time-based filter for selecting sensor observation data. They are designed to work in conjunction with a filtering node.

3.2.2.3.4.4.1 Compare Absolute Date Node

This node allows us to filter the observation data with respect to an absolute date. This filter will ensure that we select the observations that match the following condition: “obs_record_time OPERATOR absolute_date”. To set up this node, connect it to a filtering node and then fill in the values for the node's properties:

- Operator: The operator to use for comparing the record time to the given absolute date: “<, <=, =, >, >=”
- Value: The absolute date to use for the comparison. Click the calendar icon to display a date picker.

The following example (**Figure 29**) will select observations that have been recorded after a specific date/time.

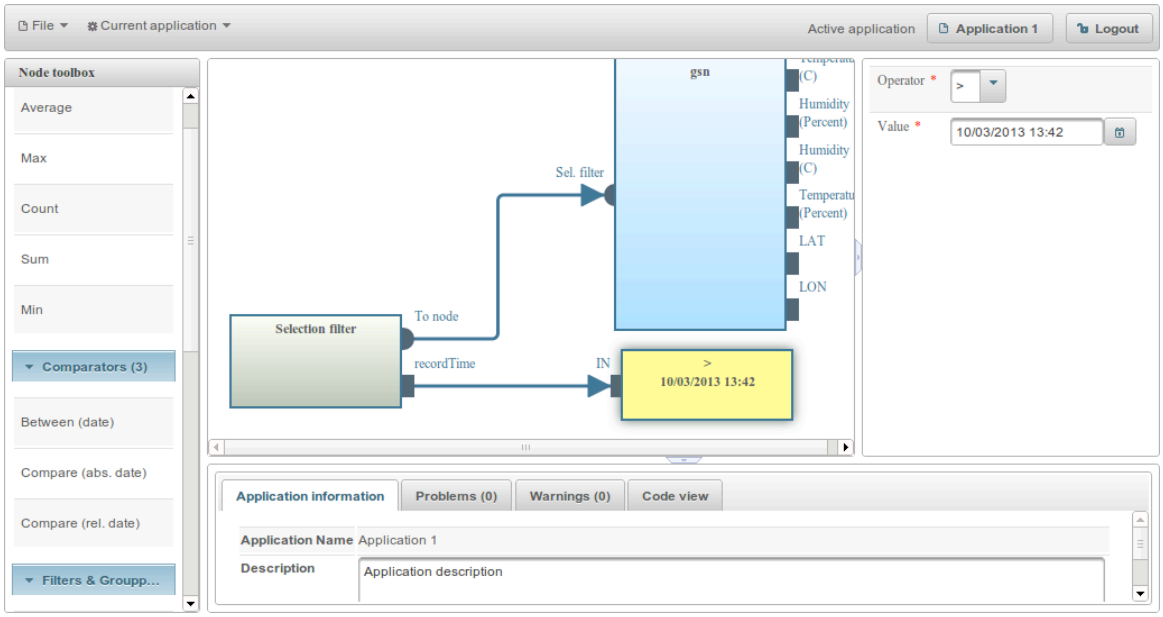

Figure 29 Request Definition compare relative date node usage

3.2.2.3.4.4.2 Compare Relative Date Node

This node allows us to filter the observation data relative to the query execution time. This filter will ensure that we select the observations that match the following condition: “NOW() - obs_record_time OPERATOR value unit”.

To set up this node, connect it to a filtering node and then fill in the values for the node's properties:

- Operator: The operator to use for comparing the record time to the given relative time unit: “<, <=, =, >, >=”
- Value: A number which will be combined with the unit to generate a comparison value.
- Unit: The comparison time unit (seconds, minutes, hours, days, months, years). It will be combined with the value to generate a comparison value.

3.2.2.3.4.4.3 Compare Between Absolute Dates Node

This node allows us to select observation data between two absolute dates. This filter is useful when we need to query historical data between two dates. Observation selection is performed using the following condition: “from_date <= obs_record_time <= to_date”

To set up this node, connect it to a filtering node and then fill in the values for the node's properties:

- From date: The earliest date to select observations from.
- To date: The latest date to select observations from.

3.3 Request Presentation

3.3.1 Main Released Functionalities & Services

The Request Presentation module is a Web application that provides end users with a visual interface to services created using the Request Definition Web application. When a user accesses the Web application, all his/her modelled applications are automatically loaded. Each application contains one or more visualization widgets. The Request Presentation layer parses the application metadata and generates a self-updating widget dashboard (see **Figure 30**).

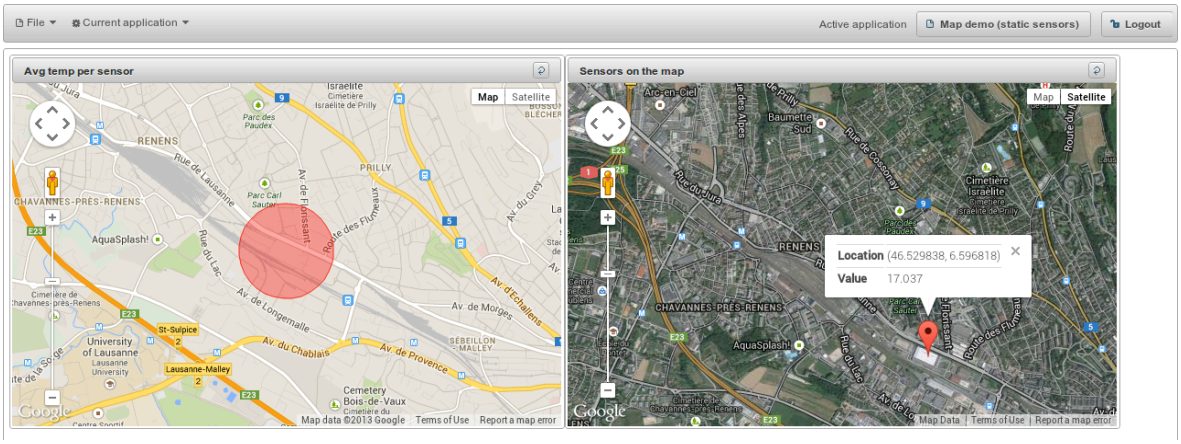

Figure 30 Request Presentation visualization widgets example

3.3.2 Download, Deploy & Run

3.3.2.1 Developer

3.3.2.1.1 System Requirements

All you need to build this project is Java 7.0 (Java SDK 1.7) or higher and Maven 3.0[9] or higher. The application this project produces is designed to be run on JBoss Enterprise Application Platform 6 or JBoss AS 7.1.

3.3.2.1.2 Download

To download Request Presentation module's source code, use your favourite git client and retrieve the code from one of the following URLs:

- HTTPS: https://github.com/OpenIotOrg/openiot.git
- SSH: git@github.com:OpenIotOrg/openiot.git

The Request Presentation module is available under the "openiot/ui/ui.requestPresentation/" folder and the Request Commons files under "openiot/ui/ui.requestCommons/".

[9] http://maven.apache.org/download.cgi#Installation

3.3.2.1.3 Deploy From the Source Code

Start JBoss Enterprise Application Platform 6 or JBoss AS 7.1 with the Web profile:

- Open a command line terminal and navigate to the root of the JBoss server directory.
- Use the following command line to start the server with the Web profile:
  - For Linux: JBOSS_HOME/bin/standalone.sh
  - For Windows: JBOSS_HOME\bin\standalone.bat

3.3.2.1.3.1 Download and Install/Deploy Dependencies

NOTE: The following build command assumes that you have configured your Maven user settings. If you have not, you must include Maven setting arguments in the command line.

- Download and deploy the Scheduler[10]
- Download and install “ui.requestCommons”
  - Open a command line terminal and navigate to the root directory of the request commons Project.
  - Type the following command to build and install:
    - “mvn clean package install”

3.3.2.1.3.2 Build and Deploy the Request Presentation Web Application

NOTE: The following build command assumes that you have configured your Maven user settings. If you have not, you must include Maven setting arguments in the command line.

- Make sure you have started the JBoss Server as described above.
- Open a command line terminal and navigate to the root directory of the Request Presentation project.
- Type this command to build and deploy the archive:
  - “mvn clean package jboss-as:deploy”

This will deploy “target/ui.requestPresentation.war” to the running instance of the server.

3.3.2.1.3.3 Access the Application

The application will be running at the following URL:
“http://servername:8080/ui.requestPresentation/”

[10] https://github.com/OpenIotOrg/openiot/wiki/Global-Scheduler

3.3.2.1.3.4 Undeploy the Archive

- Make sure you have started the JBoss Server as described above.
- Open a command line terminal and navigate to the root directory of the Request Presentation project.
- When you are finished testing, type this command to undeploy the archive:
    - "mvn jboss-as:undeploy"

3.3.2.1.4 Run in Eclipse

3.3.2.1.4.1 Integrating and Starting JBoss server

You can start JBoss Application Server and deploy the Request Presentation module from Eclipse using JBoss tools. Detailed instructions on how to integrate and start JBoss AS from Eclipse with JBoss Tools are available at the following link: https://docs.jboss.org/author/display/AS7/Starting+JBoss+AS+from+Eclipse+with+JBoss+Tools

3.3.2.1.4.2 Integrating and Deploying Request Presentation

To integrate and deploy the Request Presentation module in Eclipse one should follow the steps below:

1. Import existing maven project "File>Import>Maven>Existing Maven Projects"
2. Click the "Browse" button and navigate to the Request Presentation module's source code directory that has been previously downloaded.
3. Choose the "ui.requestPresentation" and click the "Finish" button.
4. Right click on the "ui.requestPresentation" project and select "Run As>Maven Build…"
5. Insert the following paramaters:
    a. Goals: "clean package jboss-as:deploy"
    b. Profiles: "arq-jbossas-remote"
    c. Name: "ui.requestPresentation package-deploy" (or a preferred name)
6. Click the "Run" button (the JBoss Server should be already running). The project will automatically build itself, get deployed and run at the JBoss AS running instance. From now on, this configuration should be available at the Eclipse Run Configurations under Maven Build.

To undeploy the Request Presentation module from the running instance of the JBoss AS follow the steps below:

1. Right click on the "ui.requestPresentation" project and choose "Run As>Maven Build…"
2. Insert the following parameters:
    a. Goals: "jboss-as:undeploy"
    b. Profiles: "arq-jbossas-remote"
    c. Name: "ui.requestPresentation undeploy" (or your preferred name)

Click the Run button (the JBoss Server should be already running). The project will automatically be undeployed from the JBoss AS running instance. From now on, this

configuration should be available at the Eclipse Run Configurations under Maven Build.

3.3.2.2 User

3.3.2.2.1 System Requirements

All you need to run this project is Java 7.0 (Java SDK 1.7) or higher and JBoss Enterprise Application Platform 6 or JBoss AS 7.1.
You can download the binaries through the OpenIoT Wiki[11] under the Users>Downloads[12] section.

3.3.2.2.2 Deployment/Undeployment

3.3.2.2.2.1 JBoss AS 6.0

Deploy: To deploy the Request Presentation module on JBoss AS 6.0, copy the “ui.requestPresentation.war” to the server's “deploy” directory.

Undeploy: Remove the application war file (ui.requestPresentation.war) from the JBoss deploy directory while the server is running.

3.3.2.2.2.2 JBoss AS 7.0

Deploy: To deploy the Request Presentation module on JBoss AS 7.0, copy the “ui.requestPresentation.war” to the server's “standalone/deployments” directory.

Undeploy: To undeploy the application, you need to remove the “.deployed” marker file that is generated upon successful deployment of the Request Presentation module.

You can find more detailed directions on the ins and outs of deployment on JBoss AS7 here: https://docs.jboss.org/author/display/AS7/Application+deployment

3.3.2.2.3 Manual

3.3.2.2.3.1 Login

In order to use the Request Presentation UI, a valid OpenIoT account is required. When the user first launches the Request Presentation UI they will be automatically redirected to a login screen (Figure 31). After a successful login or registration, the user will be redirected to the main application UI.

[11] https://github.com/OpenIotOrg/openiot/wiki

[12] https://github.com/OpenIotOrg/openiot/wiki/Downloads

Figure 31 Request Presentation login

3.3.2.2.3.2 The Application Menus

3.3.2.2.3.2.1 File Menu

In Figure 32, we can see the File menu.

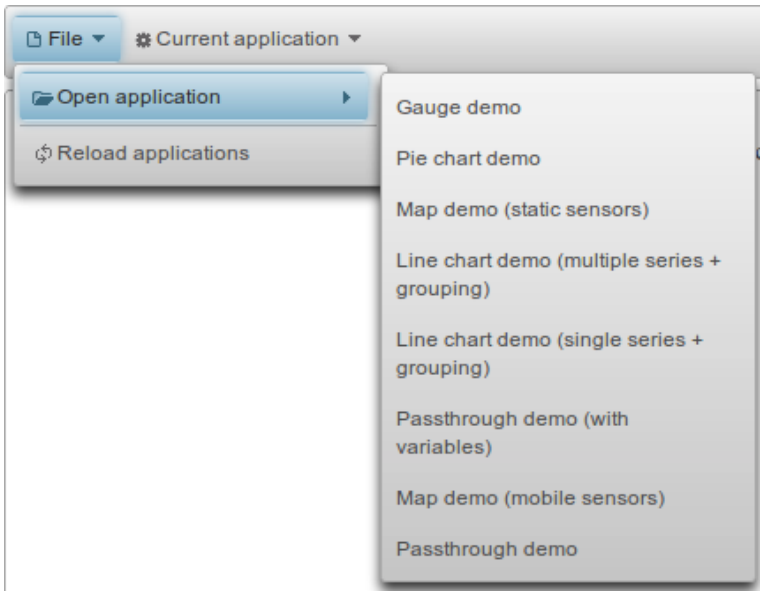

Figure 32 Request Presentation File menu

Open application

To open a specific application dashboard, click the “Open application” menu and select one of the available applications. A dashboard containing all sink nodes that were defined in the selected applications will be generated and displayed. The dashboard data will be periodically updated every 1 minute.

Reload applications

This menu option allows the user to reload their applications from the OpenIoT repositories. The loaded applications will replace the currently loaded applications.

3.3.2.2.3.2.2 The Current Application Menu

In Figure 33, we can see the “Current application” menu

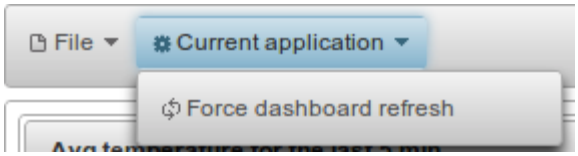

Figure 33 Request Presentation “Current application” menu

Force dashboard refresh

The dashboard data will be periodically updated every 1 minute. It is possible, however, to manually trigger a data refresh for the entire dashboard by selecting this menu item. Keep in mind that all dashboard widgets update asynchronously and independently from each other.

3.3.2.2.3.3 Visual Representation of Sink Nodes

3.3.2.2.3.3.1 Meter Gauge Node

The meter gauge node (Figure 34) displays its input with a meter gauge configured with the property values that were set by the Request Presentation UI. To reset the widget measurement, click the icon in the top-left corner of the widget.

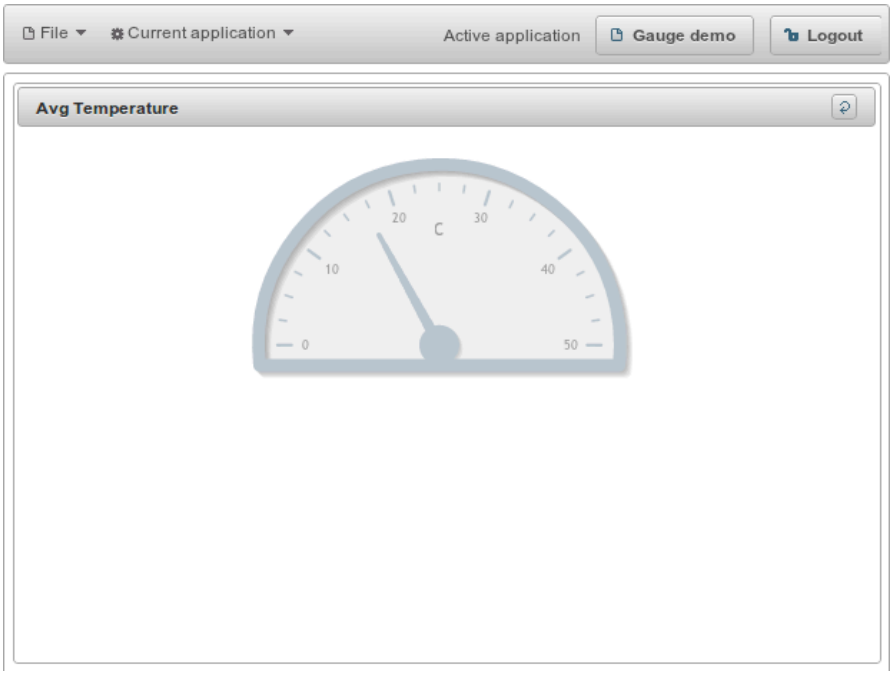

Figure 34 Request Presentation meter gauge node

3.3.2.2.3.3.2 Map Node

This node displays a map where the node's inputs are displayed using markers or circle overlays (Figure 35). To reset the map's markers and overlays, click the icon in the top-left corner of the widget.

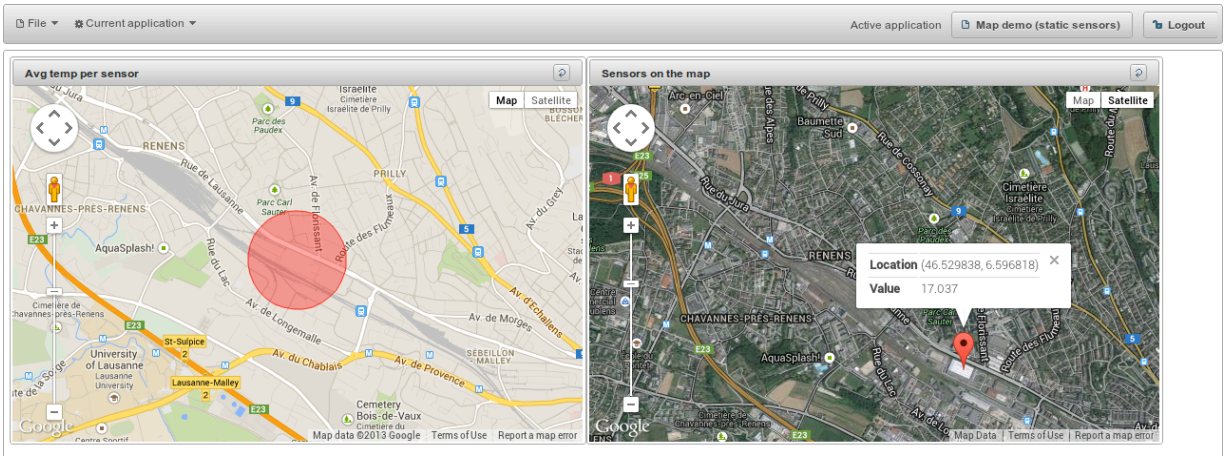

Figure 35 Request Presentation map nodes

3.3.2.2.3.3.3 Pie Chart Node

This node displays a pie chart with a configurable number of series (Figure 36). To reset the widget measurements, click the icon in the top-left corner of the widget.

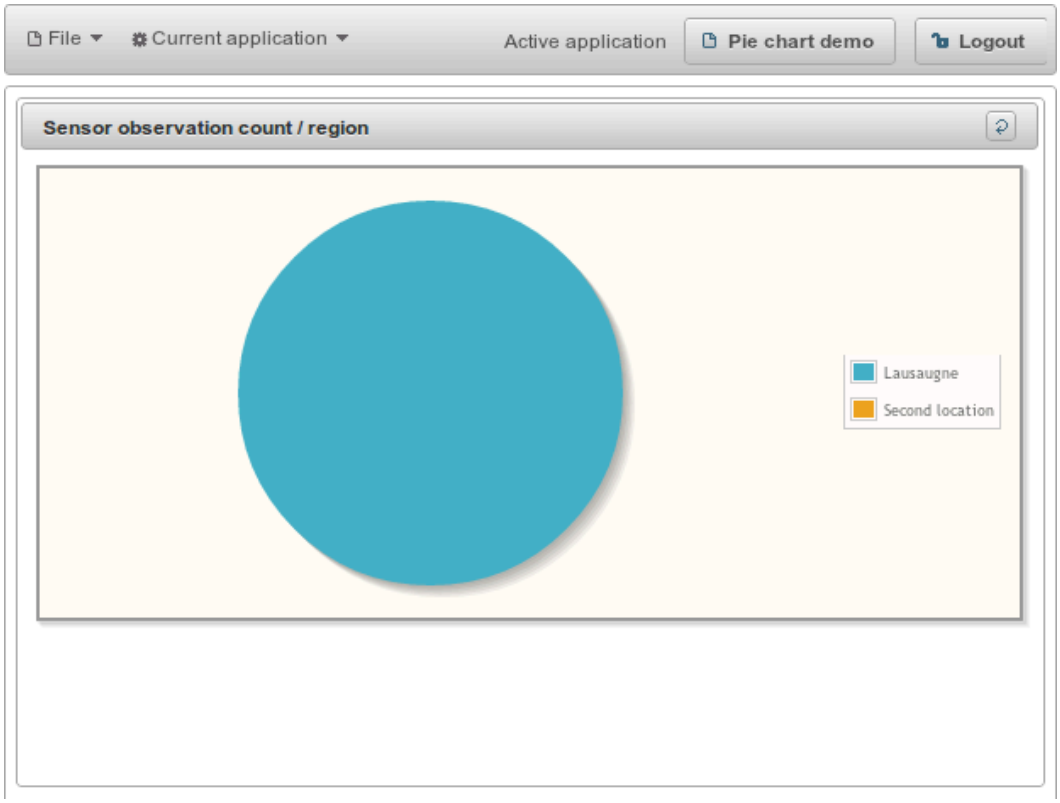

Figure 36 Request Presentation pie chart node

3.3.2.2.3.3.4 Line Chart Node

This node displays a line chart that supports multiple user-configurable series (Figure 37). You can examine a value at a particular point by hovering over the point with your mouse. To reset the widget measurements, click the icon in the top-left corner of the widget.

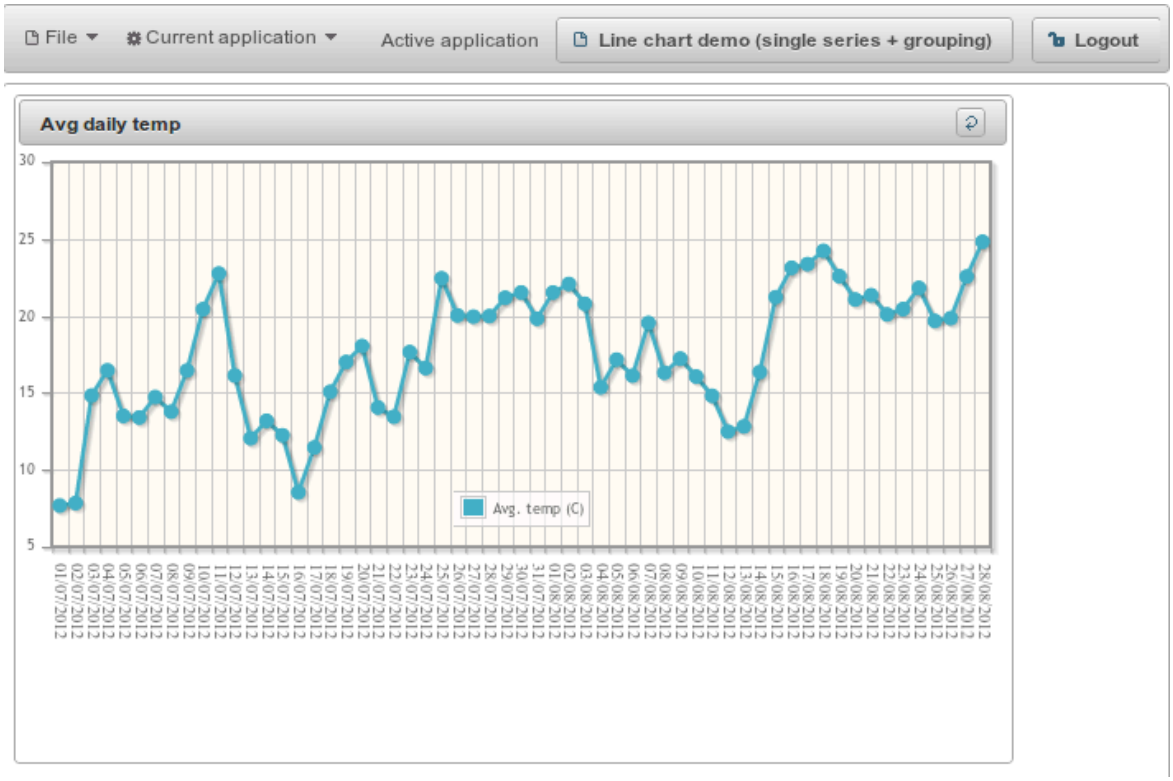

Figure 37 Request Presentation line chart node

3.3.2.2.3.3.5 Passthrough Node

This is a special node that works as a generic sink. It allows the user to define a configurable number of attribute endpoints and connect values to each one of the attributes. This widget renders all attribute values in tabular form (Figure 38). To reset the widget measurement, click the icon in the top-left corner of the widget.

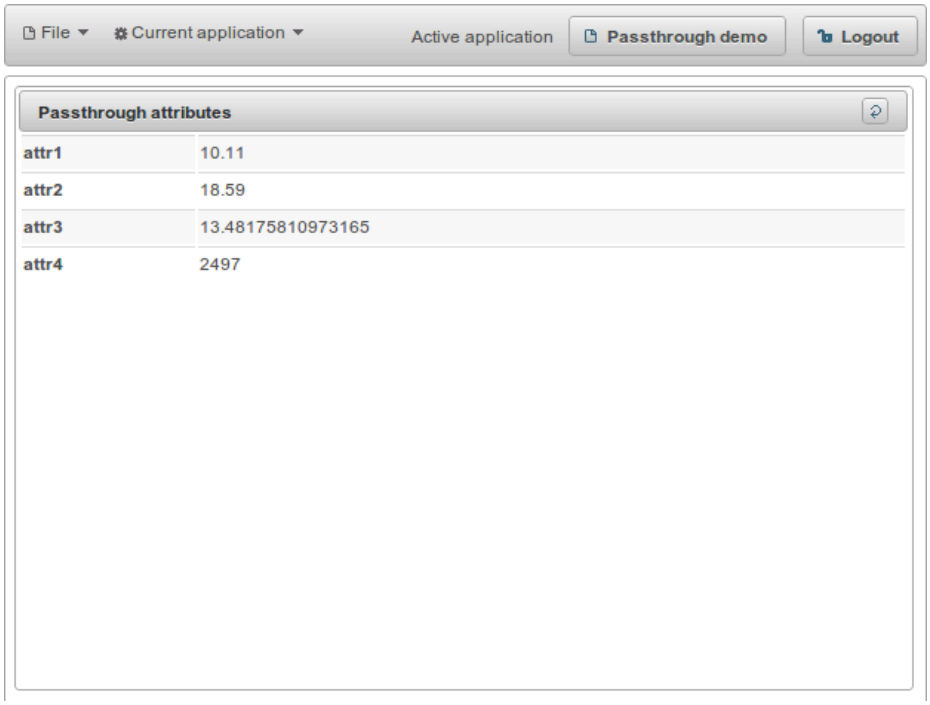

Figure 38 Request Presentation passthrough node

3.4 Sensor Schema Editor (Storage Management)

3.4.1 Main Released Functionalities & Services

The Sensor Schema Editor supports the average user in annotating sensor and sensor-related using the OpenIoT ontology and Linked Data principles. The interface automates the generation of RDF descriptions for sensor node information submitted by users.

The Sensor Schema Editor has two parts, namely the frontend interface (as depicted in **Figure 39**) and the backend LD4Sensors Server. The Schema Editor interface depends on the LD4Sensors Server for generating Linked Data descriptions for sensor metadata and sensor observations. The LD4Sensor exposes a REST API that takes as input the senor metadata as payloads and returns the RDF representation.

The Sensor Schema editor is developed using a JSF framework and deployed on JBOSS AS 7.1.1. The LD4Sensor server is a standalone server that runs the restlet framework (http://restlet.org/).

Currently, the Sensor Schema Editor is capable of generating new sensor instances based on the OpenIoT ontology. The next release of the Sensor Schema Editor will allow the creation of new sensors types by dynamically incorporating ontology extensions.

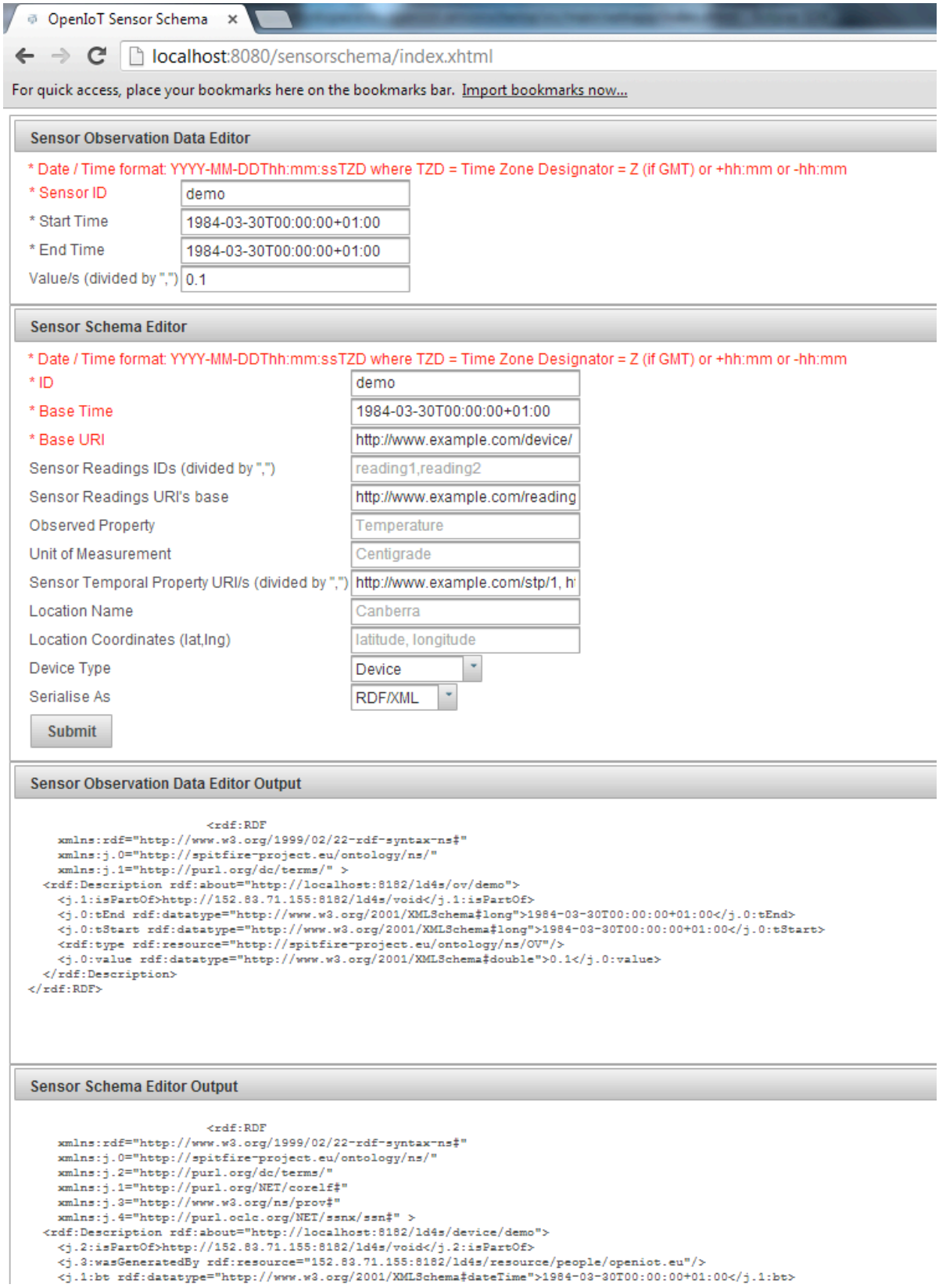

Figure 39 Sensor Schema Editor interface

3.4.2 Download, Deploy & Run

3.4.2.1 Developer

3.4.2.1.1 System Requirements

All you need to build this project is:

- Java 7.0 (Java SDK 1.7)
- JBoss AS 7.1.X
- LD4Sensor Server: The LD4Sensor server is the linked data for sensor server that converts the inputs from the user into equivalent RDF specifications. Please refer to LD4Sensor documentation on how to run the LD4Sensor server

3.4.2.1.2 Download

To download the Sensor Schema Editor UI source code, use your favourite git client and retrieve the code from one of the following URLs:

- HTTPS: https://github.com/OpenIotOrg/openiot.git
- SSH: git@github.com:OpenIotOrg/openiot.git

The source code for the Sensor Schema Editor UI is available from https://github.com/OpenIotOrg/openiot under the UI directory by the name RDFSensorSchemaEditor.

3.4.2.1.3 Deploy From the Source Code

If you have not yet done so, you must configure Maven before testing the Sensor Schema Editor deployment. After that:

- Start the JBoss Enterprise Application Platform 6 or JBoss AS 7.1 with the Web profile
  1. Open a command line terminal and navigate to the root of the JBoss server directory.
  2. Use the following command line to start the server with the Web profile:
     - For Linux: JBOSS_HOME/bin/standalone.sh
     - For Windows: JBOSS_HOME\bin\standalone.bat
- Build and deploy the RDF Sensor Schema Editor
  NOTE: The following build command assumes that you have configured your Maven user settings. If you have not, you must include Maven setting arguments in the command line.
  1. Make sure you have started the JBoss Server as described above.
  2. Open a command line terminal and navigate to the root directory of the RDF Sensor Schema Editor project.
  3. Type this command to build and deploy the archive:
     - mvn clean package jboss-as:deploy
  4. This will deploy target/sensorschema.war to the running instance of the server.

- Access the application
  - The application will be running at the following URL: http://localhost:8080/sensorschema.

- Undeploy the Archive
  1. Make sure you have started the JBoss Server as described above.
  2. Open a command line terminal and navigate to the root directory of the RDF Sensor Schema Editor Project.
  3. When you are finished testing, type this command to undeploy the archive:
     - mvn jboss-as:undeploy.

3.4.2.1.4 Run in Eclipse

You can start JBoss Application Server and deploy the RDF Sensor Schema Editor from Eclipse using JBoss tools. Detailed instructions on how to integrate and start JBoss AS from Eclipse with JBoss Tools are available at the following link: https://docs.jboss.org/author/display/AS7/Starting+JBoss+AS+from+Eclipse+with+JBoss+Tools

3.4.2.1.4.1 Integrating and Deploying RDF Sensor Schema Editor

To integrate and deploy the RDF Sensor Schema Editor in Eclipse one should follow the steps below:

1. Import existing maven project “File>Import>Maven>Existing Maven Projects”
2. Click the “Browse” button and navigate to the RDF Sensor Schema Editor module’s source code directory that has been previously downloaded.
3. Select the “RDFSensorSchemaEditor” and click the “Finish” button.
4. Right click on the “RDFSensorSchemaEditor” project and select “Run As>Maven Build…”
5. Insert the following parameters:
   a. Goals: “clean package jboss-as:deploy”
   b. Profiles: “arq-jbossas-remote”
   c. Name: “RDFSensorSchemaEditor package-deploy” (or your preferred name)
6. Click the “Run” button (the JBoss Server should be already running). The project will automatically build itself, get deployed and run at the JBoss AS running instance. From now on, this configuration should be available at the Eclipse Run Configurations under Maven Build.

To undeploy the RDF Sensor Schema Editor from the running instance of the JBoss AS follow the steps below:

1. Right click on the “RDFSensorSchemaEditor” project and select “Run As>Maven Build…”
2. Insert the following parameters:
   a. Goals: “jboss-as:undeploy”
   b. Profiles: “arq-jbossas-remote”
   c. Name: “RDFSensorSchemaEditor undeploy” (or your preferred name)

Click the “Run” button (the JBoss Server should be already running). The project will automatically be undeployed from the JBoss AS running instance. From now on, this configuration should be available at the Eclipse Run Configurations under Maven Build.

3.4.2.2 User

3.4.2.2.1 System Requirements

All you need to run this project is

- Java 7.0 (Java SDK 1.7) or higher
- JBoss Enterprise Application Platform 6 or JBoss AS 7.1.

You can download the binaries through the OpenIoT Wiki[13] under the Users>Downloads[14] section.

3.4.2.2.2 Deployment/Undeployment

3.4.2.2.2.1 JBoss AS 6.0

Deploy: To deploy the RDF Schema Editor on JBoss AS 6.0, copy the “sensorschema.war” to the server's “deploy” directory.

Undeploy: Remove the application war file (sensorschema.war) from the JBoss deploy directory while the server is running.

3.4.2.2.2.2 JBoss AS 7.0

Deploy: To deploy the RDF Schema Editor module on JBoss AS 7.0, copy the “sensorschema.war” to the server's “standalone/deployments” directory.

Undeploy: To undeploy the application, you need to remove the “.deployed” marker file that is generated upon successful deployment of the RDF Schema Editor module.

You can find more detailed directions on the ins and outs of deployment on JBoss AS7 here: https://docs.jboss.org/author/display/AS7/Application+deployment

3.4.2.2.3 Manual

Please open the following link in your favourite Web browser: http://localhost:8080/sensorschema/index.html

Once the required information is keyed in (default values are pre-filled), click the “Submit” button to obtain the sensor schema in the requested format.

[13] https://github.com/OpenIotOrg/openiot/wiki

[14] https://github.com/OpenIotOrg/openiot/wiki/Downloads

The following sections will provide a brief overview of the Sensor Schema Editor interface. The Sensor Schema Editor follows the OpenIoT ontology[15] which is derived from the Semantic Sensor Network ontology[16]

3.4.2.2.3.1 Sensor Schema Editor – Sensor Section

The Sensor Schema Editor section allows the user to define a new sensor instance based on the OpenIoT ontology.

The user can define the following fields

- Id: the identification of the actual sensor, e.g., a name of the sensor (weather station 1)
- Base Time and URI: The base time and URI refers to the sensor creation time and its uniform resource identifier used to identify the sensor resource on the Web. (In new versions, both fields will be auto-generated based on the ontology and integration with LSM.)
- Sensor Readings Ids and URI base: The id for each individual sensor reading. The combination of the URI and the Ids will allow a user to uniquely identify a sensor reading.
- Observed Property: A property the sensor observers, e.g., temperature
- Unit of Measurement: The unit of the property being observed, e.g., centigrade
- Location Name and Coordinates: Name and latitude/longitude pair for the sensor location
- Devise Type: Can be one of the following namely “Device”, “Sensing Device” or “Sensor”

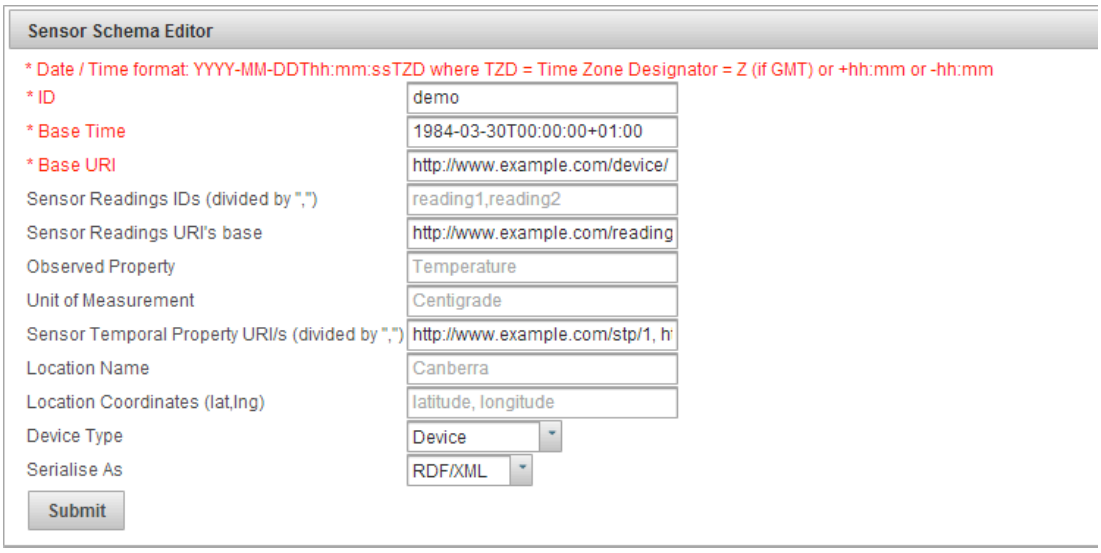

Figure 40 Sensor Schema Editor – sensor section

[15] http://openiot.eu/?q=ontology/ns

[16] http://www.w3.org/2005/Incubator/ssn/ssnx/ssn

3.4.2.2.3.2 Sensor Schema Editor - Observation Section

The observation section of the UI allows the user to create new values for the property being observed by the sensor previously defined. The user can define the following fields:

- Sensor ID: The id of the sensor that this observation value belongs to
- Start and End Time: The time period during which the observation was recorded
- Values: A range of values separated by commas.

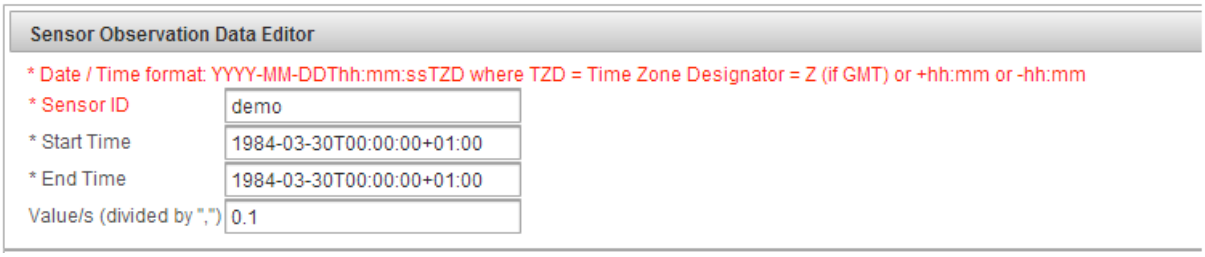

Figure 41 Sensor Schema Editor – observation section

3.5 Platform's Modules Runtime Monitoring

3.5.1 Main Released Functionalities & Services

As explained in Section 3.1, monitoring is one of the main functionalities of the OpenIoT Host Environment (IDE). The monitoring module uses the JavaMelody library to implement the relevant functionality. JavaMelody [Java Melody Project (2013)] can be configured to provide monitoring information for the entire Host Environment, as well as each individual component. More specifically, JavaMelody can [Java Melody Project (2013)]:

- provide facts about the average response times and number of executions
- make decisions when trends are bad before problems become too serious
- optimize based on the more limiting response times
- find the root causes of response times
- verify the real improvement after optimizations
- includes summary charts which can be viewed on the current day, week, month, year or custom period, showing the evolution over time of the following indicators [Java Melody Project (2013)]:

- Number of executions, mean execution times and percentage of errors of HTTP requests, SQL requests, JSF actions, JSP pages or methods of business façades (if EJB3, Spring or Guice)
- Java memory
- Java CPU
- Number of user sessions
- Number of JDBC connections

JavaMelody also includes statistics of predefined counters (current HTTP requests, SQL requests, JSF actions, JSP pages and methods of business façades for EJB3, Spring or Guice) with, for each counter [Java Melody Project (2013)]:

- A summary indicating the overall number of executions, the average execution time, the CPU time, and the percentage of errors.
- The percentage of time spent in the requests for which the average time exceeds a configurable threshold.
- The complete list of requests, aggregated without dynamic parameters with, for each, the number of executions, the mean execution time, the mean CPU time, the percentage of errors and an evolution chart of execution time over time.
- Each HTTP request indicates the size of the flow response, the mean number of SQL executions and the mean execution time.
- It also includes statistics on HTTP errors, on warnings and errors in logs, on data caches if Ehcache and on batch jobs if Quartz.
- Number of executions, mean execution times and percentage of errors of HTTP requests, SQL requests, JSF actions, struts actions, JSP pages or methods of business façades (if EJB3, Spring or Guice)

3.5.2 Download, Deploy & Run

3.5.2.1 Developer

3.5.2.1.1 System Requirements

The requirements for this project is Java 7.0 (Java SDK 1.7) or higher, Maven 3.0 or higher. The application this project produces is designed to be run on JBoss Enterprise Application Platform 6 or JBoss AS 7.1.

3.5.2.1.2 Download

JavaMelody is already declared as a dependency in the Host Environment (IDE) module's pom.xml. Therefore, no particular action is needed to download JavaMelody for the specific project.

In order to include JavaMelody as a dependency in the IDE, the following code snippet has been added to pom.xml.

```
<dependency>
      <groupId>net.bull.javamelody</groupId>
      <artifactId>javamelody-core</artifactId>
      <version>1.47.0</version>
</dependency>
```

In a non-Maven project, the JavaMelody .jar giles and the libraries dependencies may be downloaded from the following URL https://code.google.com/p/javamelody/

3.5.2.1.3 Deploy From the Source Code

As mentioned above, JavaMelody is integrated into the IDE host environment, so no separate deployment is necessary. The steps explained in Section 3.1.2.1.3 deploy the JavaMelody library along with the IDE.

3.5.2.2 User

3.5.2.2.1 System Requirements

The IDE Core supports any modern web browser (Chrome, Firefox, Safari, Opera, Internet Explorer > IE8).

3.5.2.2.2 Deployment/Undeployment

Deploy: To deploy the IDE, copy the “ide.core.war” to the server's “standalone/deployments” directory.

Undeploy: To undeploy the application, you need to remove the “.deployed” marker file that is generated upon successful deployment of the IDE module.

You can find more detailed directions for deployment on JBoss AS7 here: https://docs.jboss.org/author/display/AS7/Application+deployment

3.5.2.2.3 Manual

In **Figure 42** below, we can see the available monitoring options according to the deployed components.

Figure 42 IDE monitoring options

By clicking on a monitoring option, we get the screen depicted in **Figure 43** below.

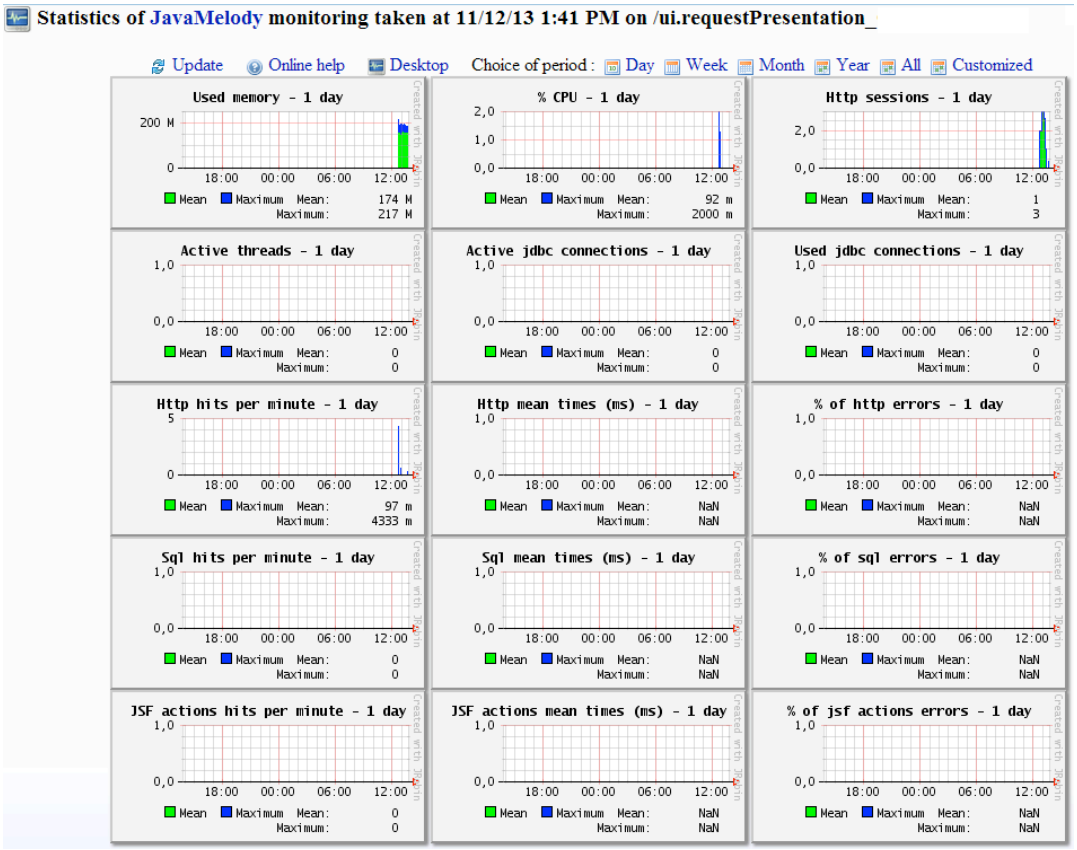

Figure 43 JavaMelody monitoring graphs

These charts (**Figure 43**) display various statistics concerning a particular module. By clicking on one of the charts we get a more detailed representation, such as the one in **Figure 44** below.

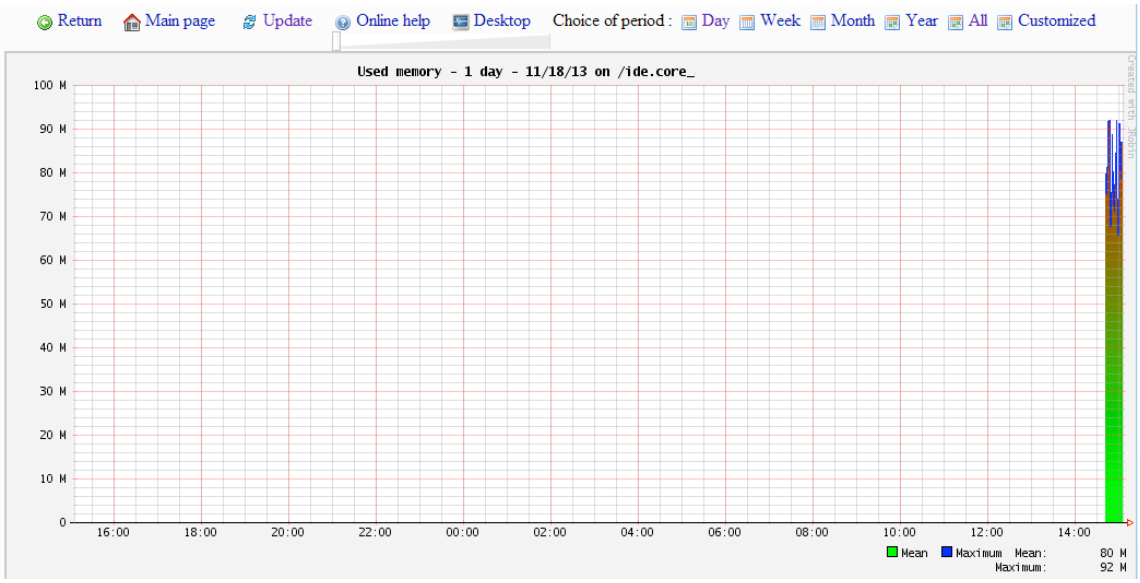

Figure 44 JavaMelody detailed monitor

Further down from the initial monitor screen, we can see various statistics collected by JavaMelody. An example in presented in **Figure 45** below.

Statistics http - 1 day

Request	% of cumulative time	Hits	Mean time (ms)	Max time (ms)	Standard deviation	% of cumulative cpu time	Mean cpu time (ms)	% of system error	Mean size (Kb)
http global	100	118	247	8,214	1,148	100	13	0.00	8
http warning	0	0	-1	0	-1	0	-1	0.00	0
http severe	66	3	6,508	8,214	2,747	46	249	0.00	8

0 hits/min on 30 requests ⊞ Details

Statistics sql - 1 day

None

Statistics jsf actions - 1 day

Request	% of cumulative time	Hits	Mean time (ms)	Max time (ms)	Standard deviation	% of cumulative cpu time	Mean cpu time (ms)	% of system error
jsf global	100	10	1	10	3	0	0	0.00
jsf warning	0	0	-1	0	-1	0	-1	0.00
jsf severe	0	0	-1	0	-1	0	-1	0.00

0 hits/min on 3 requests ⊞ Details

Figure 45 JavaMelody statistics

By clicking on the bottom right corner we can see more details concerning the various statistic group as shown in **Figure 46** below.

Request	% of cumulative time	Hits	Mean time (ms)	Max time (ms)	Standard deviation	% of cumulative cpu time	Mean cpu time (ms)	% of system error	Mean size (Kb)
http global	100	171	25	1,497	140	100	9	0.00	1
http warning	0	0	-1	0	-1	0	-1	0.00	0
http severe	70	7	440	1,497	555	72	173	0.00	5

3 hits/min on 26 requests ⊟ Details

Request	% of cumulative time	Hits	Mean time (ms)	Max time (ms)	Standard deviation	% of cumulative cpu time	Mean cpu time (ms)	% of system error	Mean size (Kb)
/home.jsf GET	70	7	440	1,497	555	72	173	0.00	5
/resources/img/openiotlogo_left_IDE.jpg GET	12	5	104	506	224	0	0	0.00	23
/home.jsf ajax POST	3	2	69	83	19	7	62	0.00	0
/welcome.jsf GET	2	5	21	38	10	3	12	0.00	1
/javax.faces.resource/style.css.jsf GET	2	7	14	36	14	3	8	0.00	1
/javax.faces.resource/theme.css.jsf GET	1	12	4	5	4	0	0	0.00	0
/ GET	1	4	12	24	10	0	0	0.00	0
/javax.faces.resource/jquery/jquery-plugins.js.jsf GET	0	7	5	23	7	0	2	0.00	0
/javax.faces.resource/primefaces.js.jsf GET	0	12	3	7	2	1	2	0.00	0
/javax.faces.resource/layout/layout.css.jsf GET	0	7	4	9	3	0	0	0.00	0
/javax.faces.resource/primefaces.css.jsf GET	0	12	2	3	1	0	1	0.00	0
/javax.faces.resource/jquery/jquery.js.jsf GET	0	12	1	3	0	0	0	0.00	0
/javax.faces.resource/layout/layout.js.jsf GET	0	7	2	5	1	1	4	0.00	0
/javax.faces.resource/images/ui-icons_f9bd01_256x240.png.jsf GET	0	6	3	6	1	0	2	0.00	0
/javax.faces.resource/images/ui-bg_inset-hard_100_fcfdfd_1x100.png.jsf GET	0	11	1	2	0	0	0	0.00	0
/javax.faces.resource/primefaces-extensions.js.jsf GET	0	7	2	3	0	0	2	0.00	0
/javax.faces.resource/images/ui-icons_469bdd_256x240.png.jsf GET	0	6	2	6	1	1	5	0.00	0
/javax.faces.resource/layout/toggle-up.gif.jsf GET	0	5	2	4	1	0	0	0.00	0
/javax.faces.resource/images/ui-bg_gloss-wave_55_5c9ccc_500x100.png.jsf GET	0	6	1	2	0	0	0	0.00	0
/javax.faces.resource/layout/toggle-lt.gif.jsf GET	0	5	1	3	0	0	0	0.00	0
/javax.faces.resource/images/ui-bg_glass_85_dfeffc_1x400.png.jsf GET	0	6	1	2	0	0	2	0.00	0
/javax.faces.resource/images/ui-bg_inset-hard_100_f5f8f9_1x100.png.jsf GET	0	6	1	3	0	0	0	0.00	0
/javax.faces.resource/images/ui-icons_217bc0_256x240.png.jsf GET	0	3	2	2	0	0	0	0.00	0
/resources/img/progress.gif GET	0	6	1	2	0	0	2	0.00	5
/javax.faces.resource/images/ui-bg_glass_75_d0e5f5_1x400.png.jsf GET	0	3	1	2	0	0	5	0.00	0
/javax.faces.resource/images/ui-icons_6da8d5_256x240.png.jsf GET	0	2	1	2	0	0	7	0.00	0

Figure 46 JavaMelody detailed statistics

Further down, we can view (**Figure 47** below) system and thread information as well as execute some JVM and Container commands such as:

- Execute the Java Garbage Collector
- Generate a Heap Dumb
- View memory histogram
- Invalidate HTTP sessions
- View HTTP sessions
- View deployment descriptor
- View MBeans
- View OS processes
- JNDI tree

Figure 47 JavaMelody system and thread information

Additionally we can view further detailed information concerning System and Threads. By clicking on “System Details”, we get the screen, depicted in **Figure 48** below, which provides us with JVM and OS-related information.

Nb of used jdbc connections (opened if no datasource):	0 ⊟ Details
OS:	Windows 7 Service Pack 1, x86/32 (2 cores)
Java:	Java(TM) SE Runtime Environment, 1.6.0_41-b02
JVM:	Java HotSpot(TM) Client VM, 20.14-b01, mixed mode
PID of process:	4076
Server:	JBoss Web/7.0.13.Final
Webapp context:	/ide.core
Start:	11/18/13 3:21 PM
JVM arguments:	-Drebel.env.ide=Eclipse -Drebel.notification.url=http://127.0.0.1:8795/jrebel/notifications -Drebel.workspace.path=C:\Users\Chris-Geo\JBoss Workspace -Drebel.log.file=C:\Users\Chris-Geo\.jrebel\jrebel.log -Drebel.properties=C:\Users\Chris-Geo\.jrebel\jrebel.properties -javaagent:C:\Users\Chris-Geo\.eclipse\org.eclipse.platform_4.3.0_1898909965_win32_win32_x86\plugins\org.zeroturnaround.eclipse.embedder_5.4.1.SR1-201311061306\jrebel\jrebel.jar -Dprogram.name=JBossTools: jboss-as-7.1.1.Final -Xms64m -Xmx512m -XX:MaxPermSize=256m -Dorg.jboss.resolver.warning=true -Djava.net.preferIPv4Stack=true -Dsun.rmi.dgc.client.gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000 -Djboss.modules.system.pkgs=org.jboss.byteman -Djava.awt.headless=true -Dorg.jboss.boot.log.file=C:/jboss-as-7.1.1.Final/standalone/log/boot.log -Dlogging.configuration=file:/C:/jboss-as-7.1.1.Final/standalone/configuration/logging.properties -Djboss.home.dir=C:/jboss-as-7.1.1.Final -Dfile.encoding=Cp1253
Mean age of http sessions (min):	16
Memory:	Non heap memory = 83 Mb (Perm Gen, Code Cache), Loaded classes = 16.414, Garbage collection time = 6.207 ms, Process cpu time = 41.948 ms, Committed virtual memory = 327 Mb, Free physical memory = 1.140 Mb, Total physical memory = 4.095 Mb, Free swap space = 3.458 Mb, Total swap space = 4.095 Mb
Perm Gen memory:	76 Mb / 256 Mb
Free disk space:	150,331 Mb

Figure 48 JavaMelody system details

Similarly, we get the screen in **Figure 49** by clicking on “Thread Details” which provides information related to the threads running on the OS.

Thread	Daemon ?	Priority	State	Executed method	Cpu time (ms)	User time (ms)	Kill
Attach Listener	yes	5	RUNNABLE		0	0	
ConnectionValidator	yes	5	TIMED_WAITING	sun.misc.Unsafe.park(Native Method)	0	0	
ContainerBackgroundProcessor[StandardEngine[jboss.web]]	yes	5	TIMED_WAITING	java.lang.Thread.sleep(Native Method)	0	0	
DeploymentScanner-threads - 1	no	5	TIMED_WAITING	sun.misc.Unsafe.park(Native Method)	78	62	
DestroyJavaVM	no	5	RUNNABLE		2,277	2,028	
Finalizer	yes	8	WAITING	java.lang.Object.wait(Native Method)	15	15	
FSWatchShutdownOnTermination	yes	5	RUNNABLE	java.lang.ProcessImpl.waitFor(Native Method)	0	0	
http-localhost-127.0.0.1-8080-1	yes	5	RUNNABLE	java.lang.Thread.dumpThreads(Native Method)	795	686	
http-localhost-127.0.0.1-8080-2	yes	5	RUNNABLE	java.net.SocketInputStream.socketRead0(Native Method)	46	46	
http-localhost-127.0.0.1-8080-3	yes	5	RUNNABLE	java.net.SocketInputStream.socketRead0(Native Method)	46	46	
http-localhost-127.0.0.1-8080-4	yes	5	RUNNABLE	java.net.SocketInputStream.socketRead0(Native Method)	31	31	
http-localhost-127.0.0.1-8080-5	yes	5	RUNNABLE	java.net.SocketInputStream.socketRead0(Native Method)	202	156	
http-localhost-127.0.0.1-8080-6	yes	5	RUNNABLE	java.net.SocketInputStream.socketRead0(Native Method)	0	0	
http-localhost-127.0.0.1-8080-7	yes	5	RUNNABLE	java.net.SocketInputStream.socketRead0(Native Method)	0	0	
http-localhost-127.0.0.1-8080-Acceptor-0	yes	5	RUNNABLE	java.net.PlainSocketImpl.socketAccept(Native Method)	0	0	
http-localhost-127.0.0.1-8080-Poller	yes	5	TIMED_WAITING	java.lang.Object.wait(Native Method)	0	0	
IdleRemover	yes	5	TIMED_WAITING	sun.misc.Unsafe.park(Native Method)	15	15	
javamelody ide.core	yes	5	TIMED_WAITING	java.lang.Object.wait(Native Method)	358	280	
jrobin ide.core	yes	5	TIMED_WAITING	java.lang.Object.wait(Native Method)	0	0	
Keep-Alive-Timer	yes	8	TIMED_WAITING	java.lang.Thread.sleep(Native Method)	0	0	
MSC service thread 1-1	no	5	WAITING	sun.misc.Unsafe.park(Native Method)	2,620	1,840	
MSC service thread 1-2	no	5	WAITING	sun.misc.Unsafe.park(Native Method)	4,867	3,135	
MSC service thread 1-3	no	5	WAITING	sun.misc.Unsafe.park(Native Method)	3,853	2,854	
MSC service thread 1-4	no	5	WAITING	sun.misc.Unsafe.park(Native Method)	5,553	4,336	
OutputReader	yes	9	TIMED_WAITING	java.lang.Thread.sleep(Native Method)	109	62	
OutputReader	yes	9	TIMED_WAITING	java.lang.Thread.sleep(Native Method)	109	62	
Periodic Recovery	no	5	TIMED_WAITING	java.lang.Object.wait(Native Method)	31	31	
rebel-debugger-thread	yes	5	TIMED_WAITING	java.lang.Thread.sleep(Native Method)	31	15	
rebel-heartbeat-thread	yes	10	TIMED_WAITING	java.lang.Thread.sleep(Native Method)	0	0	
rebel-redeploy-thread	yes	5	TIMED_WAITING	java.lang.Thread.sleep(Native Method)	0	0	
Reference Handler	yes	10	WAITING	java.lang.Object.wait(Native Method)	0	0	
Reference Reaper	yes	5	WAITING	java.lang.Object.wait(Native Method)	0	0	
Remoting "chris-geo-pc" read-1	no	5	RUNNABLE	sun.nio.ch.WindowsSelectorImpl$SubSelector.poll0(Native Method)	0	0	
Remoting "chris-geo-pc" write-1	no	5	RUNNABLE	sun.nio.ch.WindowsSelectorImpl$SubSelector.poll0(Native Method)	0	0	
Remoting "chris-geo-pc:MANAGEMENT" read-1	no	5	RUNNABLE	sun.nio.ch.WindowsSelectorImpl$SubSelector.poll0(Native Method)	234	156	

Figure 49 JavaMelody thread details

4 CONCLUSIONS

OpenIoT has released a first-of-a-kind middleware platform for the development and deployment of non-trivial IoT services, which exploit cutting edge Semantic Web technologies in a cloud environment. This platform provides a paradigm for the development of IoT/cloud applications. Such a paradigm is expected to be handy for IoT solution providers and integrators. In order to facilitate solution providers and integrators in the task of adopting and using the OpenIoT platform, the project has already provided a range of development and deployment tools. Furthermore, it has combined and bundled these tools in the scope of an integrated development environment (IDE). To the best of our knowledge, OpenIoT is the first project to provide integrated development functionalities for semantic IoT services that are delivered in a cloud-based model. Furthermore, it is probably the only open-source IDE for IoT applications and services.

The OpenIoT IDE provides several tools that facilitate developers in the task of building IoT applications and services. As a prominent example, it offers a tool that enables the definition of IoT service in a graphical manner (i.e., without programming effort) and on the basis of the dynamic selection of sensors/ICOs and the execution of operations over their data streams. As another example, the OpenIoT IDE provides tools for the visualization of data streaming from OpenIoT service. As a third example, there is also a tool facilitating the description/definition of a sensor and its subsequent semantic annotation in an OpenIoT-compliant way (i.e., based on the OpenIoT / W3SSN ontology). Furthermore, the IDE offers services for monitoring the status of sensors and services within the OpenIoT platform.

The functionalities listed in this deliverable are indicative of the OpenIoT IDE concept, while offering some attractive features that could essentially boost developers' productivity. As users and developers engage with the OpenIoT open-source project, we expect to receive significant feedback for fine-tuning and enhancing the existing tools. Furthermore, the consortium is planning the development of new tools that could showcase visual development/configuration capabilities associated with other technical aspects of the project, such as the definition of utility metrics and schemes, as well as the configuration of access control and authorization aspects. Both the improved functionalities and the new ones will be implemented and reported as part of the second version of the present deliverable, which will lead to the final version of the OpenIoT IDE as well.

5 REFERENCES

[Java Melody Project 2013], "Monitoring of JavaEE applications", retrieved November 2013 available at: https://code.google.com/p/javamelody/

OpenIoT 2013