



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

D4.3.2 Core OpenIoT Middleware Platform b

Document Id:	OpenIoT-D432-140805-Draft
File Name:	OpenIoT-D432-140805-Draft.doc
Document reference:	Deliverable 4.3.2
Version:	Draft
Editor(s):	Nikos Kefalakis, Stavros Petris, John Soldatos
Organisation:	AIT
Date:	2014-08-05
Document type:	Deliverable (Prototype)
Dissemination level:	PU (Public)

Copyright © 2014 OpenIoT Consortium: NUIG-National University of Ireland Galway, Ireland; EPFL - Ecole Polytechnique Fédérale de Lausanne, Switzerland; Fraunhofer Institute IOSB, Germany; AIT - Athens Information Technology, Greece; CSIRO - Commonwealth Scientific and Industrial Research Organization, Australia; SENSAP Systems S.A., Greece; AcrossLimits, Malta. UNIZG-FER University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia. Project co-funded by the European Commission within FP7 Program.

PROPRIETARY RIGHTS STATEMENT

This document contains information, which is proprietary to the OpenIoT Consortium.
Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to any third party, in whole or in parts, except with prior written consent of the consortium

DOCUMENT HISTORY

Rev.	Author(s)	Organisation(s)	Date	Comments
V200	Nikos Kefalakis	AIT	2014/04/09	First version based on D4.3.1
V201	Nikos Kefalakis	AIT	2014/04/09	ToC updated
V202	Nikos Kefalakis	AIT	2014/05/11	Assignments, structure modifications and other editions
V203	Nikos Kefalakis	AIT	2014/05/22	Updated chapter 3.1 (Scheduler), 3.2 (SD&UM). Added chapter 3.9 (utilities & libraries)
V204	Prem Jayaraman	CSIRO	2014/05/25	Complete Package as a Virtual Machine (section 4.4), Schema Editor requirements and use (3.8.3)
V205	Aleksandar Antonić	UNIZG-FER	2014/05/26	Added sections about the CUPUS middleware and QoS manager
V206	Nikos Kefalakis	AIT	2014/05/26	Edited chapter 2 (overview of Open source release architecture). Merged previous versions. Added chapter 7 (OpenIoT end to end demonstration)
V207	Jean-Paul Calbimonte	EPFL	2014/05/26	X-GSN description, deployment (section 3.4), and libraries (section 6.1.3). X-GSN setup (section 7.2.1)
V208	Mehdi Riahi	EPFL	2014/06/02	Security module deployment, description and libraries
V209	Nikos Kefalakis	AIT	2014/06/05	Added chapter 1.4 (updates from the first release), updated chapter 1 (introduction). Merged previous versions.

V210	Nikos Zarokostas	SENSAP	2014/06/10	Updated Request Definition and Request presentation (3.8.1, 3.8.2) and Libraries (in section 6.1.8)
V211	Hoan Nguyen Mau Quoc	DERI	2014/06/10	LSM-LIGHT modules deployment, description and libraries
V212	Nikos Kefalakis	AIT	2014/06/11	Updated chapter 4 (source code & binaries), added chapter 5 (download/deploy/run instruction). Merged previous versions
	John Soldatos	AIT	2014/06/11	Updated Introduction & Conclusion
V213	Jean-Paul Calbimonte	EPFL	2014/06/11	Remote X-GSN metadata services
V214	Hoan Nguyen Mau Quoc	DERI	2014/06/11	Corrections on “Setting up the experiment” configuration ontology
V215	Nikos Kefalakis	AIT	2014/06/23	Merged previous versions, various corrections and updates. Ready for TR
V216	Reinhard Herzog	IOSB	2014/06/25	Technical Review
V217	Prem Jayaraman	CSIRO	2014/06/30	Updated Table 22
V218	Nikos Kefalakis	AIT	2014/07/03	Addressed TR comments ready for QR
V219	Johan E. Bengtsson	AL	2014/07/14	Quality Review
V220	Nikos Kefalakis	AIT	2014/07/20	Addressed QR comments. Circulated for Approval
V221	Martin Serrano	NUIG	2014/07/25	Approved
Draft	Martin Serrano	NUIG	2014/08/05	EC Submitted

TABLE OF CONTENTS

GLOSSARY AND OPENIOT TERMINOLOGY	11
1 INTRODUCTION	13
1.1 SCOPE	13
1.2 AUDIENCE.....	14
1.3 SUMMARY.....	15
1.4 PLATFORM UPDATES FROM FIRST RELEASE (D4.3.1)	15
1.5 STRUCTURE.....	16
2 OVERVIEW OF OPEN SOURCE RELEASE.....	17
2.1 OPENIOT ARCHITECTURE	17
2.2 OPENIOT PROTOTYPE IMPLEMENTATION.....	19
3 OPENIOT COMPONENTS.....	23
3.1 SCHEDULER.....	23
3.1.1 Main Released Functionalities & Services.....	23
3.1.2 Install and Run	26
3.1.2.1 Developer.....	26
3.1.2.2 User.....	26
3.2 SERVICE DELIVERY & UTILITY MANAGER	28
3.2.1 Main Released Functionalities & Services.....	28
3.2.2 Install and Run	29
3.2.2.1 Developer.....	29
3.2.2.2 User.....	30
3.3 LINKED STREAM MIDDLEWARE LIGHT AS THE DATA PLATFORM.....	31
3.3.1 Main Released Functionalities & Services.....	31
3.3.2 Install and Run	33
3.3.2.1 Developer.....	33
3.3.2.2 User.....	37
3.4 X-GSN (EXTENDED GLOBAL SENSOR NETWORK)	38
3.4.1 Main Released Functionalities & Services.....	38
3.4.2 Install and Run	39
3.4.2.1 Developer.....	39
3.4.2.2 User.....	40
3.5 SECURITY MODULE	44
3.5.1 Main Released Functionalities & Services.....	44
3.5.1.1 Security-server	44
3.5.1.2 Security-client.....	44
3.5.1.3 Security-management.....	45
3.5.2 Install and Run	45
3.5.2.1 Developer.....	45
3.5.2.2 User.....	46

3.6	CUPUS	47
3.6.1	Main Functionalities & Services	47
3.6.2	Install and Run	48
3.6.2.1	System requirements	48
3.6.2.2	Download	48
3.6.2.3	Deploy from the source code	48
3.6.2.4	Run in Eclipse	49
3.6.2.5	Run as a Service	49
3.7	QoS MANAGER	50
3.7.1	Main Functionalities & Services	50
3.7.2	Install and Run	51
3.7.2.1	System requirements	51
3.7.2.2	Download	51
3.7.3	User	51
3.7.3.1	System requirements	51
3.7.3.2	Published Interface	52
3.8	USER INTERFACES	53
3.8.1	Request Definition	53
3.8.1.1	Main Functionalities & Services	53
3.8.1.2	Install and Run	55
3.8.2	Request Presentation	58
3.8.2.1	Install and Run	59
3.8.3	Schema Editor	61
3.8.3.1	Main Released Functionalities & Services	61
3.8.3.2	Download	61
3.8.3.3	Deploy	61
3.8.3.4	Use	62
3.9	UTILITIES & LIBRARIES	63
3.9.1	Commons library	63
3.9.1.1	Download	64
4	SOURCE CODE & BINARIES	65
4.1	SOURCE CODE AVAILABILITY	65
4.2	SOURCE CODE STRUCTURE	65
4.3	BINARIES AVAILABILITY	66
4.4	COMPLETE PACKAGE AS A VIRTUAL MACHINE	66
4.4.1	Deploying and Starting VM	66
4.4.2	Running OpenIoT Services	67
4.4.2.1	Common Setup Configuration	67
4.4.2.2	OpenIoT Property File	67
4.4.2.3	Local Deployment Property File Example	68
4.4.2.4	DERI's Infrastructure-based Deployment Property File	68
4.4.3	OpenIoT's VirtualBox End User Manual	69
4.4.4	OpenIoT's VirtualBox Developer Manual	72

5	SYSTEM INSTALATION FOR USE AND DEVELOPMENT	73
5.1	UTILITIES & PROPERTIES	73
5.1.1	“Global” Properties.....	73
5.2	DEVELOPER INSTRUCTIONS	75
5.2.1	System requirements	75
5.2.2	Download	75
5.2.3	Deploy from the source code	75
5.2.4	Run in Eclipse	76
5.2.4.1	Integrating and Starting JBoss server	76
5.2.4.2	Integrating and deploying of an OpenIoT MODULE	76
5.3	USER	77
5.3.1	System requirements	77
5.3.2	Deployment/Undeployment.....	77
5.3.2.1	JBoss AS 6.0.....	77
5.3.2.2	JBoss AS 7.0.....	77
6	PLATFORM CONTAINERS, TOOLS & LIBRARIES USED	78
6.1	OPENIoT COMPONENTS-LIBRARIES RELATION	80
6.1.1	Scheduler.....	80
6.1.2	Service Delivery & Utility Manager.....	81
6.1.3	X-GSN.....	81
6.1.4	LSM-Light.....	83
6.1.5	Security Module	83
6.1.6	CUPUS middleware	84
6.1.7	QoS manager.....	84
6.1.8	User Interface	85
6.1.8.1	Request Presentation.....	85
6.1.8.2	Request Definition.....	85
6.1.8.3	Schema Editor.....	86
7	OPENIOT END TO END DEMONSTRATION	87
7.1	DATA CAPTURING AND FLOW DESCRIPTION	87
7.2	SETTING UP THE EXPERIMENT	88
7.2.1	Edge Server setup (X-GSN)	88
7.2.1.1	Semantic annotation of sensor data	89
7.2.1.2	Registering sensors to LSM	89
7.2.1.3	Pushing data to LSM.....	90
7.3	SERVICE SETUP	91
7.3.1	Building the Request.....	91
7.4	VISUALISING THE REQUEST.....	99
8	CONCLUSIONS.....	103

APPENDIX I – SCHEMATA 104**APPENDIX II – EXTENDING X-GSN WITH NEW WRAPPERS AND VIRTUAL PROCESSING CLASSES 113**

Writing new wrappers.....	113
initialize() method.....	113
finalize()method	114
getWrapperName()method	114
getOutputFormat()method	114
Wireless Sensor Network Example	115
Webcam Example 1	115
run() method	115
Webcam Example 2	116
Data driven systems.....	116
sendToWrapper().....	117
A fully functional wrapper	117
Writing new processing classes	119
initialize() method.....	119
Dispose() method.....	119
dataAvailable() method	120

APPENDIX III – OPENIOT INSTALLATION QUICK GUIDE 121

LIST OF FIGURES

FIGURE 1: OPENIoT MAIN CORE COMPONENTS FUNCTIONAL BLOCKS.....	17
FIGURE 2: PROTOTYPE IMPLEMENTATION DATAFLOW	20
FIGURE 3: DATA ACQUISITION NETWORK IN X-GSN	38
FIGURE 4: REQUEST DEFINITION USER INTERFACE (UI)	54
FIGURE 5: REQUEST PRESENTATION UI.....	58
FIGURE 6: OPENIoT ARCHITECTURE.	88
FIGURE 7: REQUEST DEFINITION LOG-IN.	91
FIGURE 8: REQUEST DEFINITION LOADED PROFILE.....	91
FIGURE 9: NEW APPLICATION CREATION.....	92
FIGURE 10: DISCOVER SENSORS BUTTON.	92
FIGURE 11: DISCOVER SENSORS IN BRUSSELS	93
FIGURE 12: COMPARATOR (BETWEEN) PROPERTIES.	94
FIGURE 13: GROUPING OPTIONS.	94
FIGURE 14: LINE CHART PROPERTIES.....	95
FIGURE 15: VALIDATE THE DESIGN BUTTON.....	96
FIGURE 16: SPARQL SCRIPT GENERATION.....	96
FIGURE 17: LSM-LIGHT SPARQL ENDPOINT (2 WEEKS WIND CHILL IN BRUSSELS).....	97
FIGURE 18: LSM-LIGHT SPARQL ENDPOINT RESULT (2 WEEKS WIND CHILL IN BRUSSELS).	97
FIGURE 19: LSM-LIGHT SPARQL ENDPOINT (2WEEK AIR TEMP IN BRUSSELS).	98
FIGURE 20: LSM-LIGHT SPARQL ENDPOINT RESULT (2WEEK AIR TEMP IN BRUSSELS). ...	98
FIGURE 21: SAVE APPLICATION BUTTON.....	99
FIGURE 22: REQUEST PRESENTATION LOG-IN.	99
FIGURE 23: REQUEST PRESENTATION LOADED PROFILE.....	100
FIGURE 24: LOAD “WEATHERINBRUSSELS” SCENARIO.	100
FIGURE 25: SCENARIO LOADED.....	101
FIGURE 26: REFRESH THE DASHBOARD.	101
FIGURE 27: WIND CHILL VS AIR TEMPERATURE IN BRUSSELS LINE CHART.....	102
FIGURE 28: OSDSPEC SCHEMA GRAPH.....	107
FIGURE 29: OSMO SCHEMA GRAPH.....	108
FIGURE 30: SDUMSERVICERESULTSET SCHEMA GRAPH	109
FIGURE 31: SENSORTYPES SCHEMA GRAPH	111
FIGURE 32: DESCRIPTIVEIDS SCHEMA GRAPH.....	112

LIST OF TABLES

TABLE 1: LIST OF PRIMITIVES COMPRISING THE IMPLEMENTED SCHEDULER API	23
TABLE 2: IMPLEMENTED SCHEDULER API DEFINITION.....	24
TABLE 3: LIST OF PRIMITIVES COMPRISING THE OPENIoT SD&UM IMPLEMENTED API	28
TABLE 4: SERVICE DELIVERY & UTILITY MANAGER IMPLEMENTED API DEFINITION	28
TABLE 5: LIST OF PRIMITIVES COMPRISING THE OPENIoT LSM-LIGHT API	31
TABLE 6: LSM-LIGHT API SPECIFICATION	31
TABLE 7: X-GSN MAIN FUNCTIONALITIES AVAILABLE FROM THE COMMAND LINE.....	39
TABLE 8: SAMPLE METADATA FILE.....	41
TABLE 9: CLOUD BROKER'S PUBLIC METHODS AND CONSTRUCTOR	47
TABLE 10: IMPLEMENTED CLOUD BROKER API DEFINITION	47
TABLE 11: LIST OF PRIMITIVES COMPRISING THE IMPLEMENTED QoS MANAGER.....	50
TABLE 12: IMPLEMENTED QoS MANAGER API DEFINITION	50
TABLE 13: SCHEDULER LIBRARIES AND CONTAINERS.....	80
TABLE 14: SD&UM'S LIBRARIES AND CONTAINERS	81
TABLE 15: X-GSN'S LIBRARIES AND CONTAINERS.....	81
TABLE 16: LSM'S LIGHT LIBRARIES AND CONTAINERS	83
TABLE 17: SECURITY MODULE LIBRARIES AND CONTAINERS	83
TABLE 18: CUPUS MIDDLEWARE LIBRARIES AND CONTAINERS	84
TABLE 19: QoS MANAGER LIBRARIES AND CONTAINERS.....	84
TABLE 20: REQUEST PRESENTATION'S LIBRARIES AND CONTAINERS	85
TABLE 21: REQUEST DEFINITION'S LIBRARIES AND CONTAINERS	85
TABLE 22: SCHEMA EDITOR LIBRARIES AND CONTAINERS	86
TABLE 23: WEATHER METADATA FILE.....	89
TABLE 24: OSDSPEC SCHEMA	104
TABLE 25: SDUMSERVICERESULTSET SCHEMA.....	108
TABLE 26: SENSORTYPES SCHEMA	110
TABLE 27: DESCRIPTIVEIDS SCHEMA	112

TERMS AND ACRONYMS

Term	Meaning
API	Application Programming Interface
AS	Application Server
DoW	Description-of-Work
GSN	Global Sensor Network
GUI	Graphical User Interface
HTMLDSO	HyperText Markup Language Decision Support Ontology
HTTP	Hypertext Transfer Protocol
ICOs	Internet-Connected Objects
IDE	Integrated Development Environment
IERC	IoT European Research Cluster
IoTHTML	Internet of Things Hyper-Text Markup Language
JAR	Java ARchive
JAX-WS	Java API for XML Web Services
JDBC	Java Database Connectivity
JDK	Java Development Kit
JMX	Java Management Extensions
JSF	Java Server Faces
LGPL	Lesser General Public License
LSM	Linked Sensor Middleware
LSM-Light	Linked Stream Middleware Light
OAMO	OpenIoT Application Model Object
ORDBMS	Object Relational Database Management System
OSDSpec	OpenIoT Service Description Specification
OSGi	Open Service Gateway Initiative
OSMO	OpenIoT Service Model Object
PoC	Proof of Concept
POJOs	Plain Old Java Objects
QoS	Quality of Service

RDBMS	Relational Database Management System
RDF	Resource Description Format
REST	Representational State Transfer
SD&UM	Service Delivery & Utility Manager
SOA	Service Oriented Architecture
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
WAR	Web application ARchive
WP	Work Package
WSN	Wireless Sensor Networks
WYSIWYG	What You See Is What You Get
X-GSN	eExtended Global Sensor Network
XML	eXtensible Markup Language

GLOSSARY AND OPENIOT 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) Cloud	A set of computing resources enabling the delivery of IoT services over the network and based on the use of OpenIoT 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) Request Presentation	Software components that visualize the outcomes of an OpenIoT service based on the use of appropriate mashups and mashup libraries.
(OpenIoT) Scenario	A specific set of interactions between OpenIoT components serving the needs of an application.
(OpenIoT) Sensor Middleware	A part of the OpenIoT middleware platform that facilitates access to, collection and filtering of OpenIoT data streams.
(OpenIoT) Service	An IoT service deployed over the OpenIoT platform.
(OpenIoT) Service Delivery	The process of deploying and offering an OpenIoT service, after selecting the resources that are involved in the service.

(OpenIoT) Use Case	A goal or application serving needs of end users, which is implemented based on the OpenIoT platform.
Data Stream	A stream of digital information stemming from a physical or virtual sensor.
Global Scheduler	A software component that regulates how IoT services access the different resources managed by the OpenIoT platform.
Global Sensor Networks (GSN)	An open source sensor middleware platform enabling the development and deployment of sensor services with minimum-programming effort.
Linked Sensor Data	A set of Semantic Web technologies for exposing, sharing, and connecting sensor data, information, and knowledge.
Resource Discovery	The process of detecting an IoT resource (such as a sensor, service or database).
Sensor Directory	A software service which stores, organizes and provides access to information about (physical and virtual) sensors.
Sensor Discovery	The process of detecting physical and virtual sensors, as well as of the services offered by them.
Sensor Selection	The process of selecting sensors that can contribute information to a particular service.
Utility Metrics	A set of measures contributing to the overall value and performance of IoT services
Virtual Sensor	All the physical or virtual items (i.e. services, persons, sensors, GSN nodes) which provide their information through a GSN endpoint.

1 INTRODUCTION

1.1 Scope

The goal of the OpenIoT project is to provide an open source architectural blueprint for on-demand utility-based IoT applications that exploit the convergence of cloud-computing with the Internet-of-Things. The heart of this blueprint comprises a middleware framework, which facilitates service providers to deploy and monitor IoT applications in the cloud, while also enabling service integrators and end-users to access and orchestrate internet-connected objects (ICOs) and their data. The OpenIoT architecture leverages several principles of the IoT-A reference architecture¹.

OpenIoT deliverable D4.3.2 corresponds to the second release of the open source implementation of the OpenIoT middleware, which includes a wide range of components of the OpenIoT architecture. Deliverable D4.3.2 includes both the prototype middleware and the present document/report. The OpenIoT middleware is available at the GitHub infrastructure, <https://github.com/OpenIoTOrg/openiot>. Note that this second release contains major additions and updates over the developments presented in the scope of the first release of the platform as part of Deliverable D4.3.1. These updates have been driven both by the OpenIoT workplan (i.e. they were scheduled as part of the project's initial workplan) and by feedback received on the first release (including feedback from the project's open source community).

This GitHub infrastructure (established as part of WP6 of the project) contains the prototype implementation of the various components that comprise this release of the OpenIoT middleware. At the same time, this report describes the realisation of the core OpenIoT platform components in terms of their functionality, implementation details and usage. Note that the document does not intend to serve as a detailed documentation guide for IoT application developers and solution integrators. Such documentation is provided as part of the project's developers' cookbook, while relevant information is also provided on-line. Rather, this document intends to highlight the elements of OpenIoT architecture which are covered in the scope of the second OpenIoT middleware release, while also providing information about supported functionalities.

In particular, the deliverable describes the implementation status of key elements of the OpenIoT architecture, including the service utility and delivery manager, extended GSN middleware (X-GSN), project's data platform where data are kept in a cloud environment, as well as various user interfaces and human-machine interfaces that support the user interaction with the OpenIoT platform. For each of these components the deliverable illustrates the functionalities that have been implemented, along with basic instructions for using/running the component based on source code available within the Github.

¹ <http://www.iot-a.eu/public/public-documents/d1.2>

In addition to detailing the functionality and implementation status of all components, this deliverable reports also on the wider infrastructures needed to deploy and run the OpenIoT platform. They include third-party libraries, platforms, containers and tools, which are the prerequisites for deploying and using the OpenIoT middleware framework. Furthermore, the deliverable provides some details on the status of the open source code, including availability information.

Note that this deliverable consolidates and integrates results from several other work packages of the project. Specifically, it implements the OpenIoT architecture which was specified in WP2 (Requirements, Use Cases and Technical Specifications), while it integrates the OpenIoT ontology and edge server architecture towards interfacing to internet connected objects specified in WP3 (Edge Intelligence and Interaction Protocols). Furthermore, the open source implementation builds upon infrastructures, concepts and decisions established in the scope of WP6 (Open Source Implementation and Proof-of-Concept Validation). Therefore, the deliverable has an integrating role towards achieving the main goal of the project.

1.2 Audience

The target audience for this deliverable includes:

- **OpenIoT project members**, notably members of the project that intend to engage in the deployment and/or use of the OpenIoT open source middleware framework. For these members the deliverable could serve as a valuable guide for the installation, deployment, integration and use of the various modules that comprise the OpenIoT software.
- **The IoT open source community**, which should view the present deliverable as an extensible (sophisticated) middleware for integrating IoT applications, notably applications that adopt a cloud/utility-based model. Note also that members of the open source community might be willing to contribute to the OpenIoT project too. For these members, the deliverable can serve as a basis for understanding the technical implementation of the components that comprise the second release of the OpenIoT middleware.
- **IoT researchers at large**, who could find in this deliverable a practical guide on the main elements/components that comprise a non-trivial IoT solution, notably a solution that blends cloud computing with IoT.
- **IERC projects and their members**, who could read in this deliverable the details of a practical instantiation of the IoT-A / IERC reference architecture. As already outlined, the OpenIoT architecture is largely based on the IERC reference architecture.

All the above groups could benefit from reading the report, but also from using the released prototype implementation. Note that comments and feedback received from some of the above groups on the first release of the OpenIoT middleware platform have driven several of the developments that are illustrated in this deliverable. This is for example the case with the OpenIoT Virtual Box packaging and associated utilities, which has been created in response to the project's community demand for easier installation and deployment of OpenIoT.

1.3 Summary

This report presents various components that comprise the OpenIoT middleware implementation. The report provides the structure and implementation details for all OpenIoT components, along with practical information on their usage. In addition, the deliverable includes also a wider perspective on various components within the OpenIoT architecture, given that this architecture provides the main structuring principles that guide the integration of the various components. Hence, the deliverable starts with an overview of the OpenIoT architecture in the scope of the second open source release of the project.

The deliverable makes special references to the third-party components that are needed to deploy, run and leverage the capabilities of the released open source platform. Furthermore, a concrete example of the potential use of the middleware framework is provided, as a means to illustrate the practical capabilities of this open source release.

1.4 Platform Updates from first release (D4.3.1)

In the second release of the OpenIoT platform, new modules were developed and numerous bug-fixes and enhancements were contributed from the OpenIoT consortium. Some of the most important ones are described in the list below:

- New modules:
 - QoS manager (CUPUS) development and integration into the platform (section 3.6).
 - Security module development (CAS) and integration into the platform (section 3.5).
 - Schema Editor & LD4S redesign development and integration into the platform for handling sensor types (section 3.8.3).
- Enhancements & bug-fixes:
 - LSM-light support for deployment within JBoss.
 - Scheduler update service functionality.
 - Added “feature of interest” support to LSM-Light and X-GSN so that a User will be able to bind third party data with the sensor.
 - Added ability at X-GSN to support RDF format sensor metadata at configuration time.
 - All of the modules running within JBoss adopted a “global properties” file that resides at the JBoss instance to ease configuration and deployment of the platform (section 5.1.1).
 - Hardcoded properties and various properties used within every project in various modules are now set in the global properties file.
 - Sensor type support in LSM-Light.

- New LSM-client integration, that supports sensor types in several modules like X-GSN, Scheduler and SD&UM.
- LSM-Light provided as a service from DERI premises currently for testing purpose.
- Support and deployment of LSM-Light to the cloud (Amazon).
- Added Maven support to all OpenIoT modules.
- Packaging of the complete OpenIoT platform (for use and development) to an Oracle Virtual Box for distribution (section 4.4).

A more detailed list of all enhancements and bug-fixes implemented in the OpenIoT platform can be found at the OpenIoT issue tracker² at the closed issues³ section.

1.5 Structure

The deliverable is structured as follows:

- Chapter 2 following this introduction positions the OpenIoT middleware platform and its components in the context of the OpenIoT architecture. In particular, the section presents the OpenIoT architecture in terms of its main components and their interactions.
- Chapter 3 is devoted to the presentation of various components within the second release of the core platform, in terms of their functionality and use. For several components (where applicable) the section presents also options for extending and/or enhancing their functionalities.
- Chapter 4 describes the source code of the open source release in terms of structure and availability. Moreover it provides details on the packaging and offering of the OpenIoT platform within an Oracle Virtual Box container.
- Chapter 5 provides directions for developers and users on how to download, deploy and run the OpenIoT platform modules separately.
- Chapter 6 presents the third party components and libraries that are needed in order to deploy and run various OpenIoT components. The third-party components include platforms, containers and tools.
- Chapter 7 provides an OpenIoT platform demonstration by presenting a simple scenario and its required step by step configuration of the platform and services, and its visualization.
- Chapter 8 concludes the report.
- Annexes I, II and III with relevant information about OpenIoT project Data Schema, procedures for extending OpenIoT capabilities with new wrappers and virtual processing classes and a quick guide on how to install OpenIoT.

² <https://github.com/OpenIoTOrg/openiot/issues>

³ <https://github.com/OpenIoTOrg/openiot/issues?page=1&state=closed>

2 OVERVIEW OF OPEN SOURCE RELEASE

2.1 OpenIoT Architecture

OpenIoT Architecture is comprised of nine main elements as depicted in Figure 1. The Sensor Middleware, the Pub-Sub server, the Cloud Data Storage, the Security module, the Scheduler in conjunction with Discovery Services functionality, the Service Delivery and Utility Manager, the Request Definition, the Request Presentation and the Configuration and Monitoring block. The main core components have been introduced first in D2.2⁴ and D2.3⁵, and then described with more detailed service architecture functional blocks in D4.1⁶. In this section an overview of those components with accurate refinements in functionality is included.

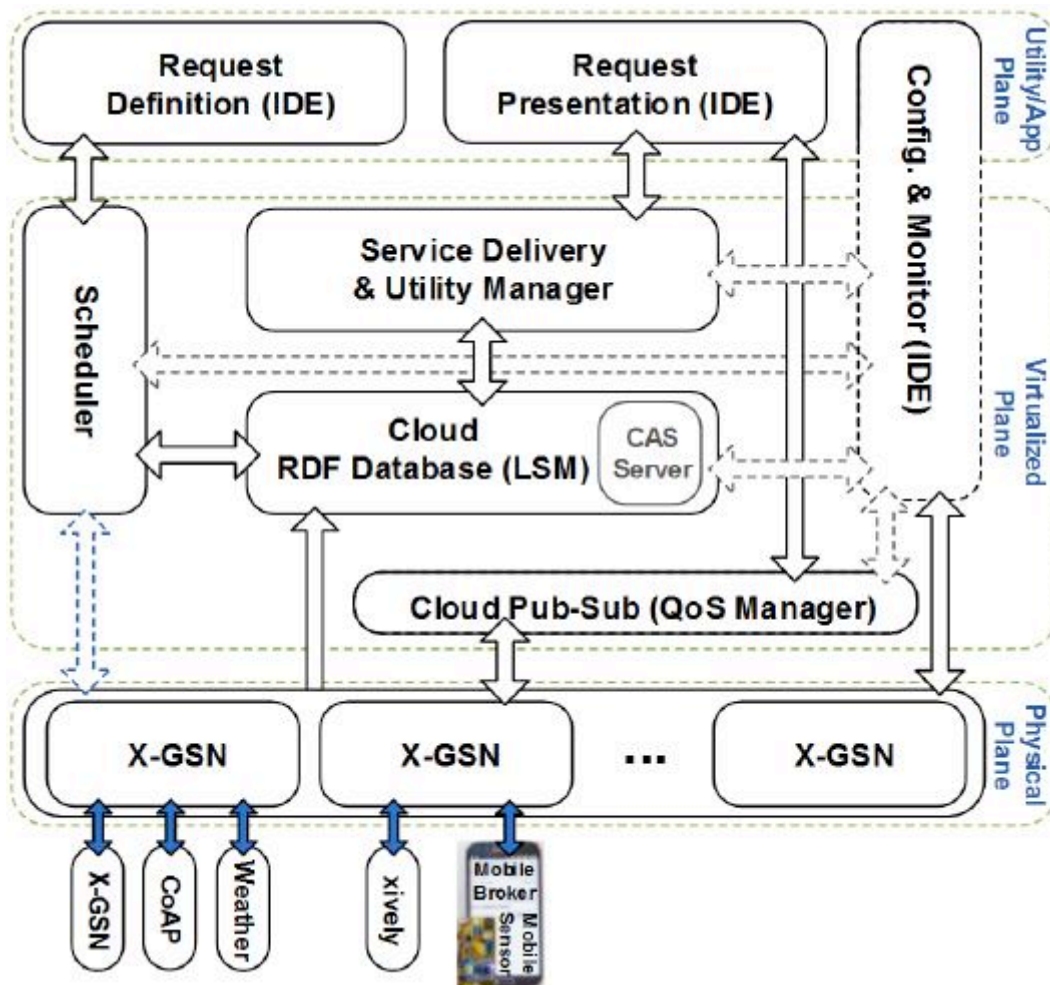


Figure 1: OpenIoT Main Core Components Functional Blocks.

⁴ D41 Service Delivery Environment Formulation Strategies

⁵ D22 OpenIoT Platform Requirements and Technical Specifications

⁶ D23 OpenIoT Detailed Architecture and Proof-of-Concept Specifications

- The **Sensor Middleware** (Extended Global Sensor Network, X-GSN) collects, filters and combines 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 implementation of the OpenIoT platform uses the GSN sensor middleware that has been extended and called X-GSN (Extended GSN).
- The **Pub-Sub** server (QoS Manager) is the component which monitors over time the global demand for sensor data generated by MIOs and manages the data acquisition process from MIOs to achieve a desired sensing coverage while optimising parameters such as energy and bandwidth consumption, sensor trustworthiness and/or data propagation latency.
- 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 OpenIoT platforms (functional data). The implementation of the OpenIoT platform uses the LSM Middleware, which has been re-designed with push-pull data functionality and cloud interfaces for enabling additional cloud-based streaming processing.
- The **Security** module (CAS server) is based on Jasig CAS⁷ and provides OAuth2.0 authentication and authorization for all other OpenIoT modules.
- The **Scheduler** processes all the requests for on-demand deployment of services 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 the service provisioning.
- 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 a SPARQL query provided by the Scheduler). To this end, this component makes use of the service description and resources identified 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.
- The **Request Definition** component enables on-the-fly specification of service requests to the OpenIoT platform. It comprises a set of services for specifying and formulating such requests, while also submitting them to the Scheduler. This component may be accompanied by a GUI (Graphical User Interface).

⁷ <http://www.jasig.org/cas>

- The **Request Presentation** component is in charge of the visualization of the outputs of a service that is provided within the OpenIoT platform. This component selects mashups from a library in order to facilitate service presentation composition. It is expected that service integrators implementing/integrating solutions with the OpenIoT platform will enhance or even override the functionality of this component for creating a GUI pertaining to their solution.
- The **Configuration and Monitoring** component enables management and configuration of functionalities of the sensors and the (OpenIoT) services that are deployed within the OpenIoT platform. It is also supported by a GUI.

2.2 OpenIoT Prototype Implementation

The OpenIoT project provides a prototype implementation which presents a set of components that demonstrates the basic workflows of services in the OpenIoT architecture. This implementation is available through the OpenIoT open source portal. The objectives of this implementation are the following:

- To provide an integrated version of the OpenIoT software to the open source community.
- To involve users, from outside the consortium and get feedback from the users/developers regarding the OpenIoT architecture.
- To bootstrap the OpenIoT open source community.

In addition, the prototype implementation will enable the identification of issues, and problems that may arise during platform usage. The prototype implementation includes the set of modules and services described above.

Figure 2 below depicts the OpenIoT prototype implementation by providing a complete example of the platform's data and service flow from during deployment, configuration and presentation stages.

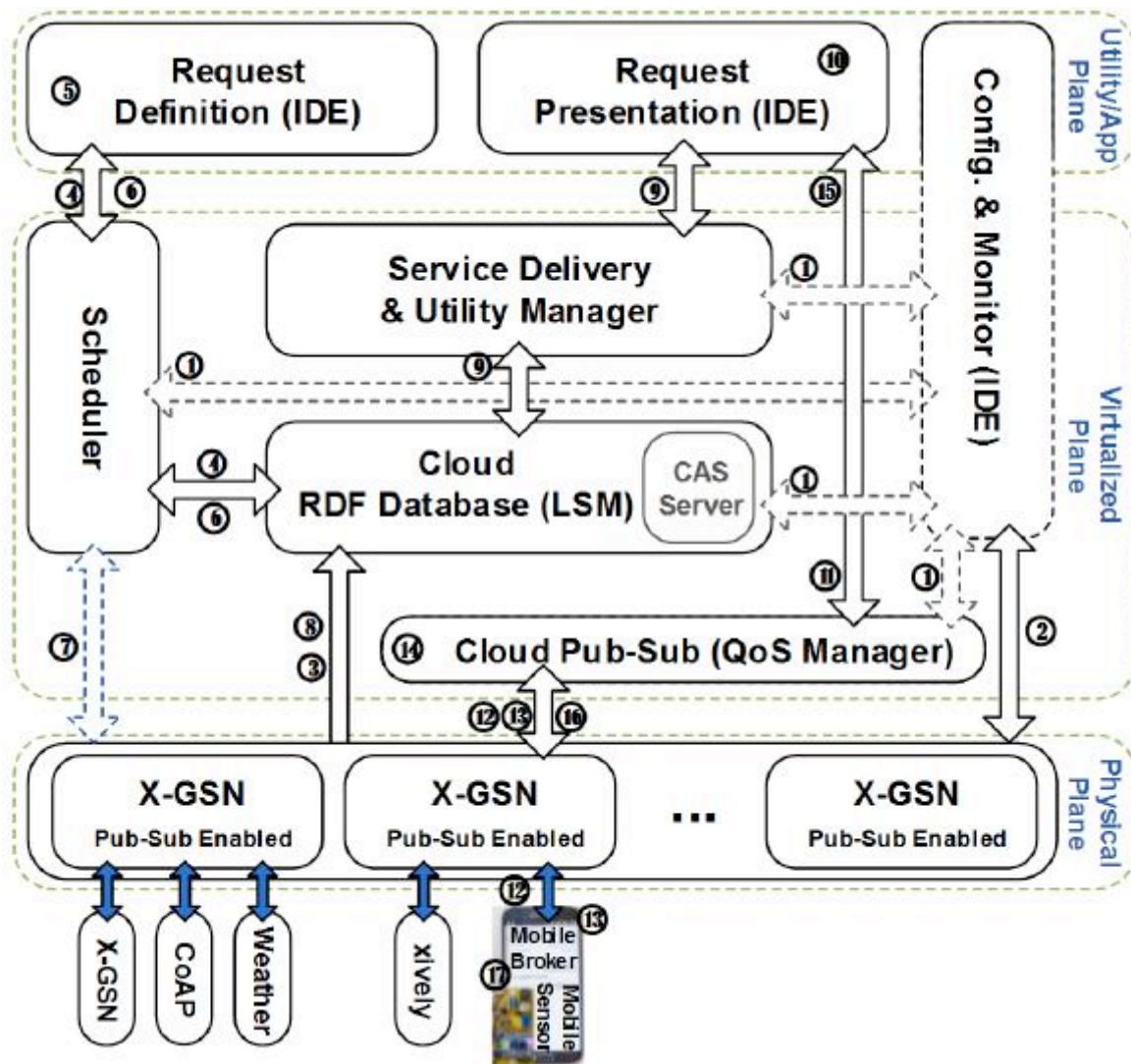


Figure 2: Prototype implementation dataflow

The data flow can be outlined as follows:

1. Module Monitoring & Management

- Platform instances monitoring is possible by using the monitoring UI which is part of the OpenIoT IDE (more details can be found in D4.4.2).

2. Sensor Configuration and Deployment

- The sensors configuration and deployment can be facilitated by using the Schema Editor.
- The generated schema can be “fed” to X-GSN so as to describe the metadata of the deployed sensor.

3. Publish Available Sensors

- X-GSN nodes “announce” the available virtual sensors to the Directory Service and start to publish their data in SSN compliant RDF format to

the “virtualSensorsDataStorage” entity based on each X-GSN local configuration.

4. Sensor Types Discovery

- A User requests from the Scheduler by using the Request Definition UI to retrieve from the Directory Service all the available sensor models that satisfy specific attributes (coordinates and radius). The request is sent to the Scheduler’s “discoverService” service.
- The Scheduler executes a combination of queries (SPARQL) to fulfil the user specified query provided by the previous step.
- The Directory Service retrieves the data (the “availableSensors” entity) and replies back to the Scheduler with the available sensor models.
- The reply is forwarded to the Request Definition UI from the Scheduler and the retrieved information is provided to the User.

5. Service Definition

- The User, with the help of Request Definition UI, defines the request by implementing rules, provided by the tool, over the reported sensor models. This information along with execution and service presentation preferences is placed in an OSDSpec (OpenIoT Service Description Specification) object (Figure 28, Figure 29).

6. Service Registration

- This OSDSpec, from the step above, is pushed to the Scheduler with the help of “registerService”.
- The Scheduler analyses the received OSDSpec and sends the request to the Directory Service (“serviceDescription” entity).

7. Indirect Sensor Activation

- X-GSN is capable of checking LSM-Light for the list of sensors currently used. If the sensor belongs in this list it can start populating LSM-Light with data.

8. Data Production

- Data are populated from X-GSN to LSM-Light in RDF form following the OpenIoT ontology.

9. Data Retrieval of Registered Service

- After having configured the request, the User is able to use the Request Presentation UI for visualizing a registered Service’s data.
- With the help of SD&UM’s “getAvailableAppIDs” the Request Presentation retrieves all the registered applications/services related to a specific User (available at the “serviceDescription” entity).
- Having selected a service, the User requests to retrieve the results related to it. This is done by submitting a “pollForReport” from the Request Presentation to the SD&UM having the applicationID as input.

- The SD&UM requests (SD&UM's "getService") from the Directory Service to retrieve all related information for the specific Service (available at the "serviceDescription" entity).
- The Directory Service provides this information to the SD&UM.
- The SD&UM analyses the retrieved information, which are available in the replied OSMO object (Figure 29), and forwards the included SPARQL script (created by the Request Definition UI and stored by the Scheduler) to the Directory Service SPARQL interface.

10. Data Visualization

- The result is sent from the Directory Service to the SD&UM, in a SparqlResultsDoc⁸ format. Then the SD&UM forwards it to the Request Presentation within a SdumServiceResultSet (Figure 30 below) object that also includes information on how these data should be presented.

11. User Subscription

- If mobile sensors are used the User can subscribe for specific data/sensors at the Pub-Sub module.

12. Direct Sensor Activation

- The mobile sensors that users are subscribed to are activated from the Pub-Sub module.

13. Real Time Data Streaming

- Data populated from the mobile devices are captured in near real time and pushed to the subscribed users/devices. In parallel they are pushed through X-GSN to LSM-Light storage.

14. Real Time Data Filtering

- Data produced from the mobile devices are filtered in near real time at the point of delivery to the platform based on the needs of the service.

15. Real Time Data Presentation

- Data are presented in near real time to the subscribed devices.

16. Mobile Sensor Data Adaptation

- The mobile data are adapted and mapped to the OpenIoT ontology.

17. Mobile Sensor Data Exchange

- The mobile device is capable of exchanging data with hardware external sensors that are bound with it.

⁸ <http://www.w3.org/TR/rdf-sparql-XMLres/#defn-srd>

3 OPENIOT COMPONENTS

This section is devoted to the presentation of the components comprising the prototype implementation release of the core OpenIoT platform that enable a user/developer to download, install and use the modules of OpenIoT platform. Since the OpenIoT platform will keep evolving over time, an updated version of the information provided in this section will be provided regularly at the OpenIoT Wiki⁹ space under the Documentation¹⁰ section.

3.1 Scheduler

The Scheduler formulates a request based on user inputs (Request Definition UI). It parses each service request and interacts accordingly with the rest of the OpenIoT platform through the Cloud Database (DB).

3.1.1 Main Released Functionalities & Services

The current release of the OpenIoT Scheduler implements the functionalities/capabilities that are reflected in the interface listed in Table 1 below.

Table 1: List of primitives comprising the implemented Scheduler API

```
<<interface>>
SchedulerInterface
---
discoverSensors (userID:String, longitude:double, latitude:double, radius:float): SensorTypes
registerApp(osdSpec: OSDSpec): String
unregisterApp(String applicationID): void
updateApp(osdSpec: OSDSpec): void
getApplication(applicationID: String): OAMO
getService (serviceID: String): OSMO
getAvailableAppIDs (userID: String): DescriptiveIDs
getAvailableServiceIDs (applicationID: String): DescriptiveIDs
getAvailableApps (userID: String): OSDSpec
getUser (userID: String): OpenIoTUser
```

Service descriptions as well as their inputs and outputs are listed in Table 2 below.

⁹ <https://github.com/OpenIoTOrg/openiot/wiki>

¹⁰ <https://github.com/OpenIoTOrg/openiot/wiki/Documentation>

Table 2: Implemented Scheduler API definition

Service Name	Input	Output	Info
discoverSensors	String userID, double longitude, double latitude, float radius	SensorTypes	Used to help applications build a request by using existing sensor classes. Requires as input UserID in String format, the location longitude/ latitude and radius specifying an area of interest. Returns a SensorTypes object which includes all available sensors in the specified area with their metadata.
registerApp	OSDSpec osdSpec	String	Used to register/submit the constructed service to the cloud. Requires as input the OpenIoT Service Description Specification (OSDS) which includes all the User's preferences regarding the Service, request lifecycle and visualization preferences. It returns the constructed Application ID.
unregisterApp	String applicationID	void	Used to unregister/delete a registered/running service. Requires as input the Application ID.
updateApp	osdSpec: OSDSpec	void	Used to update a registered service. Requires as input an OSD Specification.
getService	String serviceID	OSMO	Used to retrieve the description (OSMO) of an available service. Requires as input a Service ID.

getApplication	String applicationID	OAMO	Used to retrieve the description (OAMO) of an available Application. Requires as input the Application ID
getAvailableAppIDs	String userID	DescriptiveIDs	Used to retrieve the available applications (a list of applicationID/ServiceName/ServiceDescription triplet) already registered by a specific user. Requires as input a User ID.
getAvailableServiceIDs	String serviceID	DescriptiveIDs	Used to retrieve the available services (a list of serviceID/ServiceName/ServiceDescription triplet) already registered by a specific user. Requires as input the Service ID.
getAvailableApps	String userID	OSDSpec	Used to retrieve the services defined by a user. It returns an OpenIoT Service Description Specification. Requires as input a User ID.
getUser	String userID	OpenIoTUser	Used to retrieve user information for implementing access control mechanisms.

The Schemata as well as the schema diagrams of OSDSpec, OAMO, OSMO, SensorTypes and DescriptiveID can be found in “APPENDIX I – SCHEMATA” annexes section below in this document.

3.1.2 Install and Run

3.1.2.1 Developer

3.1.2.1.1 System requirements

All you need to build this project is:

- Java 7.0 (Java SDK 1.7) or later,
- Maven 3.0 or later
- LSM-Light client library (openiot/modules/lsm-light/lsm-light.client) installed to your local maven repository, and
- utils.commons library (openiot/utils/utils.commons/) installed to your local maven repository.

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 Scheduler's source code use your favourite git client and retrieve the code from one of the following URLs:

- HTTPS: <https://github.com/OpenlotOrg/openiot.git>
- SSH: <git@github.com:OpenlotOrg/openiot.git>

The scheduler is available under the "openiot /modules /scheduler/" folder.

Directions on how to build and run the project from the source code can be found in chapter 5 "System Instalation for Use and Development".

3.1.2.2 User

3.1.2.2.1 System requirements

To run this project you require Java 7.0 (Java SDK 1.7) or later and JBoss Enterprise Application Platform 6 or JBoss AS 7.1.

You can download the binaries through the OpenIoT Wiki ¹¹ under the Users>Downloads¹² section.

Directions on how to deploy the project from its binaries can be found in chapter 5 "System Instalation for Use and Development".

¹¹ <https://github.com/OpenlotOrg/openiot/wiki>

¹² <https://github.com/OpenlotOrg/openiot/wiki/Downloads>

3.1.2.2.2 Published Interface

This module is expected to be used from the OpenIoT Request Definition user interface. Instructions on how to install and use this interface can be found in Section “3.5.1 Request Definition”. In case you would like to use a third party application use the restful web services from the URLs listed below (the inputs and the outputs of these services are defined in Table 2 above):

- Welcome Message listing the available services:
 - <http://localhost:8080/scheduler.core/rest/services/>
- Discover Sensors :
 - <http://localhost:8080/scheduler.core/rest/services/discoverSensors>
- Registering a new service:
 - <http://localhost:8080/scheduler.core/rest/services/registerService>
- Unregister Service :
 - <http://localhost:8080/scheduler.core/rest/services/unregisterApp>
- Update Service:
 - <http://localhost:8080/scheduler.core/rest/services/updateApp>
- Get Application:
 - <http://localhost:8080/scheduler.core/rest/services/getApplication>
- Get Service:
 - <http://localhost:8080/scheduler.core/rest/services/getService>
- Get User:
 - <http://localhost:8080/scheduler.core/rest/services/getUser>
- Get Available Application IDs:
 - <http://localhost:8080/scheduler.core/rest/services/getAvailableAppIDs>
- Get Available Service IDs :
 - <http://localhost:8080/scheduler.core/rest/services/getAvailableServiceIDs>
- Get Available Applications :
 - <http://localhost:8080/scheduler.core/rest/services/getAvailableApps>

3.2 Service Delivery & Utility Manager

The Service Delivery & Utility Manager has a dual functionality. On the one hand (as a service manager), it is the module enabling data retrieval from the selected sensors comprising the OpenIoT service. On the other hand, the utility manager maintains and retrieves information structures regarding service usage and supports metering, charging and resource management processes.

3.2.1 Main Released Functionalities & Services

The current release of the OpenIoT Service Delivery & Utility Manager implements the functionalities/capabilities that are reflected in the interface listed in Table 3 below.

Table 3: List of primitives comprising the OpenIoT SD&UM implemented API

```
<<interface>>
SDUManagerInterface
---
pollForReport (applicationID: String): SdumServiceResultSet
getApplication(applicationID: String): OAMO
getService (serviceID: String): OSMO
getAvailableAppIDs (userID: String): DescriptiveIDs
getAvailableServiceIDs (applicationID: String): DescriptiveIDs
getUser (userID: String): OpenIoTUser
```

The services description as long as their inputs and outputs are listed Table 4 below.

Table 4: Service Delivery & Utility Manager implemented API definition

Service Name	Input	Output	Info
pollForReport	String serviceID	SdumServiceResultSet	Invokes a previously defined Service having the specified applicationID. This call will produce only one Result Set.
getService	String serviceID	OSMO	Used to retrieve the description (OSMO) of an available service. Requires as input a Service ID.
getApplication	String applicationID	OAMO	Used to retrieve the description (OAMO) of an available Application. Requires as input an Application ID.
getAvailableAppIDs	String userID	DescriptiveIDs	Used to retrieve the available applications (a list of

			applicationID/ServiceName/ServiceDescription triplet) already registered by a specific user. Requires as input a User ID.
getAvailableServiceIDs	String serviceID	DescriptiveIDs	Used to retrieve the available services (a list of serviceID/ServiceName/ServiceDescription triplet) already registered by a specific user. Requires as input a Service ID.
getUser	String userID	OpenIoTUser	Used to retrieve the user's information for implementing access control mechanisms.

The Schemata as well as the schema diagrams of SdumServiceResultSet, OAMO, OSMO and DescriptiveID can be found in "APPENDIX I – SCHEMATA" section below.

3.2.2 Install and 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 later,
- Maven 3.0 or later
- LSM-Light client library (openiot/modules/lsm-light/lsm-light.client) installed to your local maven repository, and
- utils.commons library (openiot/utils/utils.commons/) installed to your local maven repository.

The application this 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 SD&UM'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 SD&UM is available under the "openiot/modules/sdum/" folder.

Directions on how to build and run the project from the source code can be found in chapter 5 "System Installation for Use and Development"

3.2.2.2 User

3.2.2.2.1 System requirements

To run this project you require Java 7.0 (Java SDK 1.7) or later and JBoss Enterprise Application Platform 6 or JBoss AS 7.1.

You can download the binaries through the OpenIoT Wiki¹³ under the Users>Downloads¹⁴ section.

Directions on how to deploy the project from its binaries can be found in chapter 5 “System Instalation for Use and Development”

3.2.2.2.2 Published Interface

This module is expected to be used from the OpenIoT Request Presentation user interface. Directions on how to install and use this interface can be found in Section “3.8.2 Request Presentation”. In case you would like to use a third party application to invoke SD&UM services, use restful web services at the URLs listed below (the inputs and outputs of the services are described in Table 4 above):

- Welcome message listing the available services:
 - <http://localhost:8080/sdum.core/rest/services/>
- Poll for Report :
 - <http://localhost:8080/sdum.core/rest/services/pollforreport>
- Get Service Status:
 - <http://localhost:8080/sdum.core/rest/services/getServiceStatus>
- Get Application:
 - <http://localhost:8080/sdum.core/rest/services/getApplication>
- Get Service:
 - <http://localhost:8080/sdum.core/rest/services/getService>
- Get User:
 - <http://localhost:8080/sdum.core/rest/services/getUser>
- Get Available Application IDs:
 - <http://localhost:8080/sdum.core/rest/services/getAvailableAppIDs>
- Get Available Service IDs :
 - <http://localhost:8080/sdum.core/rest/services/getAvailableServiceIDs>

¹³ <https://github.com/OpenIoTOrg/openiot/wiki>

¹⁴ <https://github.com/OpenIoTOrg/openiot/wiki/Downloads>

3.3 Linked Stream Middleware Light as the Data Platform

Linked Stream Middleware Light (LSM-Light) is a platform that brings together the live real world sensed data and the Semantic Web. The implementation of the OpenIoT platform uses the LSM Middleware, which has been re-designed with push-pull data functionality and cloud interfaces for enabling additional cloud-based streaming processing.

An LSM deployment is available at <http://lsm.deri.ie/>. It provides functionalities such as 1) Wrappers for real time data collection and publishing; 2) A web interface for data annotation and visualization; and 3) A SPARQL endpoint for querying unified Linked Stream Data and Linked Data. The first and third functionality are the ones used in the proof-of-concept implementation in OpenIoT.

3.3.1 Main Released Functionalities & Services

In order for LSM-Light to support stream data processing programmatically, a Java API is provided. By using this API, a developer can add, delete and update GSN-generated sensor data into the implemented LSM-Light Server (triple store). Table 5 below illustrates the main API primitives that provide the LSM-Light functionalities, while Table 6 provides more details about all services that comprise the API.

Table 5: List of primitives comprising the OpenIoT LSM-Light API

```
<<interface>>
LSMServerInterface
---
getSensorById(sensorID:String): Sensor
getSensorBySource(sensorSource:String): Sensor
sensorAdd(newSensor:Sensor): void
sensorDataUpdate(observation:Observation): void
sensorDataUpdate(triples:String): void
deleteTriples(graphURL:String,triples:String): void
deleteTriples(graphURL:String): void
```

Table 6: LSM-light API Specification

Service Name	Input	Output	Info
getSensorById	String sensorID	Sensor	Used to retrieve an existing sensor from LSM by sending a request. Requires as input a sensorID, in String format, which is a unique value to identify the sensor. Returns a Sensor object that includes all the available metadata describing the sensor.

getSensorBySource	String sensorSource	Sensor	Used to retrieve an existing sensor from LSM by sending a request. Requires as input a sensorSource in String format. Returns a Sensor object that includes all the available metadata.
sensorAdd	Sensor sensor	void	Used to register a new sensor into LSM. Requires as input a Sensor class instance. This method returns a notification and sensorId indicating whether the sensor was successfully added or not.
sensorDataUpdate	Observation observation	void	Used to update the latest observed data generated by a sensor. Requires as input an Observation object that includes all the available observed data. This method returns a notification indicating whether the observed data was successfully updated or not.
deleteTriples	String graphURL	void	Used to clear all the triple data of a specific graph. Requires as input the graphURL. This method returns a notification indicating whether the data were successfully removed or not. Note that the data cannot be restored after this method is called.
deleteTriples	String graphURL, String triples	void	Used to clear specific triples from a specific graph. Requires as input the graphURL and triple patterns.

3.3.2 Install and Run

3.3.2.1 Developer

3.3.2.1.1 System requirements

LSM-LIGHT server requires Java JDK 1.7. In order to setup LSM-LIGHT modules, there are some required components that need to be installed on your local system.

3.3.2.1.1.1 VIRTUOSO SERVER

Virtuoso Universal Server is a middleware and database engine hybrid that combines the functionality of a traditional RDBMS, ORDBMS, virtual database, RDF, XML, free-text, web application server and file server functionality in a single system. For OpenIoT, we use the open source edition of Virtuoso Universal Server (it is also known as OpenLink Virtuoso).

If you want to install a virtuoso server instance in your local server, please follow the instructions available at: <http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/>

3.3.2.1.1.2 JBOSS SERVER

All you need to build this project is Java 7.0 (Java SDK 1.7) or later, Maven 3.0 or later and LSM-Light client library installed to your local maven repository. The application that 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

Source code can be cloned using GIT from OpenIoT's repository on GitHub at the following address:

<https://github.com/OpenlotOrg/openiot.git>

The LSM-LIGHT module can be found under modules/lsm-light.

3.3.2.1.3 Configuring OpenIoT properties file for LSM-LIGHT.server

In order to deploy LSM-LIGHT server, please provide all initial parameters in the OpenIoT properties file (utils/utils.commons/src/main/resources/properties/openiot.properties). The connection pool class will read all the configuration values from this file to connect to your triple storage server.

Property Name	Description	Example value
lsm-light.server.connection.driver_class	Java JDBC class name	virtuoso.jdbc4.Driver
lsm-light.server.connection.url	Server host	jdbc:virtuoso://localhost:1111/
lsm-light.server.connection.username	Username connection	dba (default)
lsm-light.server.connection.password	Password connection	dba (default)
lsm-light.server.minConnection	Minimum number of connection	5

lsm-light.server.maxConnection	Maximum number of connection	10
<i>lsm-light.server.acquireRetryAttempts</i>	After attempting to acquire a connection and failing, try to connect these many times before giving up.	5

3.3.2.1.4 Deploy from source code

LSM-LIGHT.server depends on utils.common and lsm-light.client modules. So they must be available in the local Maven repository in order to be able to build the LSM server module. For deploying the lsm-light.server module run the following command in the “openiot/modules/lsm-light/lsm-light.server” folder:

```
mvn jboss-as:deploy
```

Detailed directions on how to build and run the project from the source code can be found in chapter 5 “System Installation for Use and Development”

3.3.2.1.5 LSM-Light Sensors Data Manipulation

The examples below illustrate some functionalities of the LSM-Light API (lsm-light.client module). This shows the users how to communicate with the LSM server (lsm-light.server module) and publish their sensor data into it.

Example 1 - How to add a new sensor into the LSM triple store

```
import java.util.Date;

import org.openiot.lsm.beans.Place;
import org.openiot.lsm.beans.Sensor;
import org.openiot.lsm.beans.User;
import org.openiot.lsm.http.LSMTripleStore;
import org.openiot.lsm.utils.ObsConstant;

public class TestLSM {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

        try{
            /*
             * add new sensor to lsm store. For example: Air quality sensor from
            Lausanne
             * Sensor name: lausanne_1057
             */
        }
    }
}
```

```

// 1. Create an instance of Sensor class and set the sensor metadata
Sensor sensor = new Sensor();
sensor.setName("lausanne_1057");
sensor.setAuthor("sofiane");
sensor.setSourceType("lausanne");
sensor.setInfor("Air Quality Sensors from Lausanne");
sensor.setMetaGraph("http://lsm.deri.ie/OpenIoT/demo/sensormeta#");
sensor.setDataGraph("http://lsm.deri.ie/OpenIoT/demo/sensordata#");
sensor.setSensorType("weather");
sensor.setTimes(new Date());

//set observed properties of sensor
sensor.addProperty(ObsConstant.TEMPERATURE);
sensor.addProperty(ObsConstant.HUMIDITY);

// set sensor location information (latitude, longitude, city, country,
continent...)
Place place = new Place();
place.setLat(46.529838);
place.setLng(6.596818);
sensor.setPlace(place);

/*
 * Set sensor's author
 * If you don't have LSM account, please visit LSM Home page
(http://lsm.deri.ie) to sign up
 */

// create LSMTripleStore instance
LSMTripleStore lsmStore = new LSMTripleStore();

//call sensorAdd method
lsmStore.sensorAdd(sensor);

}catch (Exception ex) {
    ex.printStackTrace();
    System.out.println("cannot send the data to
server");
}
}
}

```

Example 2 - How to update sensor data within the LSM triple store

```

/*
 * An Observation is a Situation in which a Sensing method has been used to
estimate or
 * calculate a value of a Property of a FeatureOfInterest.
 */

//create an Observation object
Observation obs = new Observation();
obs.setDataGraph("http://lsm.deri.ie/OpenIoT/demo/sensordata#");
obs.setMetaGraph("http://lsm.deri.ie/OpenIoT/demo/sensormeta#");
// set SensorURL of observation

```

```

//for example:
"http://lsm.der.i.e/resource/8a82919d3264f4ac013264f4e14501c0" is the sensorURL of
lausanne_1057 sensor
obs.setSensor("http://lsm.der.i.e/resource/8a82919d3264f4ac013264f4e14501c0
");
//set time when the observation was observed. In this example, the time is
current local time.
obs.setTimes(new Date());
/*
 * Relation linking an Observation to the Property that was observed
 */
ObservedProperty obvTem = new ObservedProperty();
obvTem.setObservationId(obs.getId());
obvTem.setPropertyType(ObsConstant.TEMPERATURE);
obvTem.setValue(9.58485958485958);
obvTem.setUnit("C");
obs.addReading(obvTem);

ObservedProperty obvC0 = new ObservedProperty();
obvC0.setObservationId(obs.getId());
obvC0.setPropertyType(ObsConstant.HUMIDITY);
obvC0.setValue(0.0366300366300366);
obvC0.setUnit("C");
obs.addReading(obvC0);
obs.setDataGraph("http://lsm.der.i.e/OpenIoT/new/sensordata#");
obs.setMetaGraph("http://lsm.der.i.e/OpenIoT/new/sensormeta#");

lsmStore.sensorDataUpdate(obs);

```

If the sensor metadata or sensor data are already in N-Triple format, you can use these methods **sensorAdd(String triples)** and **sensorDataUpdate(String triples)** to insert directly data into LSM triple store.

Example 3 - Retrieve sensor object by sensorURL id or by sensor source

```

/*
 * the sensorURL id and sensor source are unique
 */
Sensor sensor1 =
lsmStore.getSensorById("http://lsm.der.i.e/resource/8a82919d3264f4ac013264f4e14501c0
","http://lsm.der.i.e/OpenIoT/demo/sensormeta#");
Sensor sensor2 =
lsmStore.getSensorBySource("http://opensensedata.epfl.ch:22002/gsn?REQUEST=113&name=
lausanne_1057,","http://lsm.der.i.e/OpenIoT/demo/sensormeta#");

```

Example 4 - Delete sensor data

```

/**
 * remove sensor
 * @param sensorURL
 */
lsmStore.sensorDelete("http://lsm.der.i.e/resource/8a82919d3264f4ac013264f4e145
01c0","http://lsm.der.i.e/OpenIoT/demo/sensormeta#");

/**
 * delete all reading data of specific sensor

```

```
    * @param sensorURL
    */
    lsmStore.deleteAllReadings("http://lsm.derii.ie/resource/8a82919d3264f4ac013264f4e14501c0");

    /**
     * delete sensor data at a certain period of time
     * @param sensorURL
     * @param dateOperator
     * @param fromTime
     * @param toTime
     * fromDate, toDate are java Date objects
     */
    Date fromDate = new Date();
    lsmStore.deleteAllReadings("http://lsm.derii.ie/resource/8a82919d3264f4ac013264f4e14501c0", "<=", fromDate, null);

    lsmStore.deleteAllReadings("http://lsm.derii.ie/resource/8a82919d3264f4ac013264f4e14501c0", null, fromDate, toDate);
```

3.3.2.2 User

3.3.2.2.1 System requirements

All you need to build this project is Java 7.0 (Java SDK 1.7) or later, and Maven 3.0.

3.3.2.2.2 Download

To download the source code of the lsm-light.client module, use your favourite git client and retrieve the code from one of the following URLs:

- HTTPS: <https://github.com/OpenlotOrg/openiot.git>
- SSH: <git@github.com:OpenlotOrg/openiot.git>

The lsm-light.client module is available under the “openiot/modules/lsm-light/lsm-light.client” folder.

3.3.2.2.3 Deploy and run

The lsm-light.client module provides the required API for the developers to access the lsm-light.server and perform access control. In particular, it is used by the Scheduler, SDUM and X-GSN. The LSM-LIGHT.server also depends on the utils.common module. Therefore, it must be available in the local Maven repository in order to be able to build the lsm-light.client module. In order to build and install the lsm-light.client module in the local Maven repository, run the following command in the “openiot/modules/lsm-light/lsm-light.client” folder:

```
mvn install
```

Detailed directions on how to build and install the project from the source code can be found in chapter 5 “System Instalation for Use and Development”

3.4 X-GSN (Extended Global Sensor Network)

3.4.1 Main Released Functionalities & Services

X-GSN is based on the GSN (Global Sensors Network) middleware to which it adds semantic capabilities. X-GSN is designed to facilitate the deployment and programming of sensor networks. It runs on one or more computers composing the backbone of the acquisition network (see Figure 3).

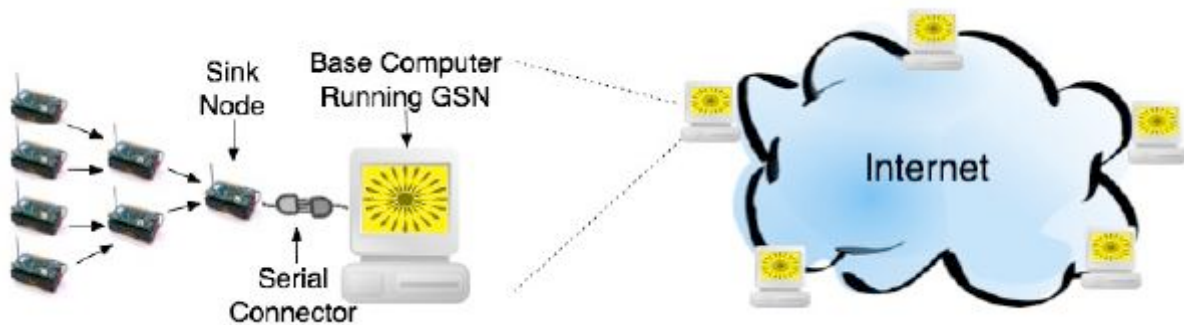


Figure 3: Data acquisition network in X-GSN

A set of wrappers allows feeding raw data into the system. Wrappers are used to encapsulate the data received from the data source into the standard X-GSN data model, called a *Stream Element*. A Stream Element is an object representing a row in the data store of X-GSN. Each wrapper is implemented as a Java class. Usually, a wrapper initializes a specialized third-party library in its constructor. It also provides a method which is called each time the library receives data from the monitored device. This method will extract the interesting data, optionally parse it, and create one or more Stream Element(s) with one or more columns. From this point on, the received data has been mapped into a SQL data structure with fields that have a name and type. X-GSN is then able to filter this using its own SQL-like syntax.

Data streams are processed according to XML specification files. The system is built upon a concept of sensors (real sensors or virtual sensors), which are data sources created from live data, and connected together in order to build the required processing path. For example, one can imagine a thermometer that would send its data into X-GSN through a wrapper, then that data stream could be sent to an averaging node, the output of this node could then be split and sent to a database for recording and to a web site for displaying the average measured temperature in real time. The described example can be realised by editing only a few XML files in order to connect the various nodes together.

3.4.2 Install and Run

3.4.2.1 Developer

3.4.2.1.1 System requirements

X-GSN requires Java JDK 1.6 or higher to run. If you use the Maven build file, this will be added automatically through the compiler plugin.

X-GSN relies on a number of other libraries but all are specified in the Maven dependencies. In particular, it has an embedded web server based on Jetty.

3.4.2.1.2 Download

Source code can be cloned using GIT from OpenIoT's repository on GitHub at the following address:

<https://github.com/OpenlotOrg/openiot.git>

X-GSN can be found under modules/x-gsn.

3.4.2.1.3 Deploy from source code

X-GSN relies on Maven. If you have not yet done so, you must configure Maven before attempting deployment.

Once Maven is correctly configured, you can generate the X-GSN jar file by invoking the "package" phase, as follows:

```
mvn package
```

This phase automatically runs all the necessary phases for preparing X-GSN to be executed including the following procedures:

- Compile
- Binding XML beans descriptions to Java through JiBX
- Generating JAR file
- Copying all dependencies from local Maven repository to target folder in order to be included in the class path for the execution of X-GSN.

Once all JAR files are in place, you can run X-GSN with the following commands provided in Table 7.

Table 7: X-GSN main functionalities available from the command line.

Command	Description
gsn-start	Starts the X-GSN server
gsn-stop	Stops the X-GSN server
lsm-register	Registers sensor metadata to LSM

The X-GSN web server will be running and listening on default port 22001. You can see it running by browsing the page at: <http://localhost:22001>

3.4.2.2 User

3.4.2.2.1 System requirements

X-GSN requires Java JDK 1.6 or 1.7 to run.

3.4.2.2.2 Download

X-GSN (xgsn.jar) can be downloaded as a Java JAR from the OpenIoT binary distribution repository¹⁵.

3.4.2.2.3 Deployment

In order to run X-GSN, you run the JAR installer from command line or through the GUI (if jar files are correctly associated with JAVA). From command line, you can run the following command:

```
java -jar xgsn.jar
```

The wizard will ask you about the home folder to create for X-GSN. It may also ask for the location of the Java JDK if it is not able to find it.

3.4.2.2.4 Use

3.4.2.2.4.1 RUNNING X-GSN

X-GSN can be controlled from the command line with the commands provided in **Table 7** above.

3.4.2.2.4.2 SEMANTIC ANNOTATION OF SENSOR DATA

3.4.2.2.4.2.1 CONFIGURING GSN FOR INTEGRATION WITH LSM

In order to connect GSN with LSM, you need to edit the file **application.conf**. There, you need to specify user credentials and LSM schema to use. See the following sample:

```
username=gsnuser
password=gsnpass
metaGraph="http://lsm.der.i.e/OpenIoT/sensormeta#"
dataGraph="http://lsm.der.i.e/OpenIoT/sensordata#"
lsm.server="http://localhost:8080/lsm-light.server/"
```

¹⁵ <https://github.com/OpenIoTOrg/openiot/wiki/Downloads>

3.4.2.2.4.2 CREATING VIRTUAL SENSOR METADATA

In order to associate metadata with a virtual sensor, you need to create a metadata file that will be associated with the virtual sensor. The metadata file has to be located in the same folder as the virtual sensor and needs to have the same name as the virtual sensor name appended with the extension (**.metadata**). For example, a virtual sensor named `sensor1.xml` will have an associated metadata file named `sensor1.metadata`. Sample virtual sensors with associated metadata can be found in (**virtual-sensors/samples/LSM**).

Table 8 below shows a sample metadata file.

Table 8: Sample metadata file

```
sensorID="http://lsm.der.i.e/resource/61330620147099"
sensorName=opensense_1
source="http://planetdata.epfl.ch:22002/gsn?REQUEST=113&name=opensense_1"
sourceType=lausanne
sensorType=lausanne
information=Air Quality Sensors from Lausanne station 1
author=opensense
feature="http://lsm.der.i.e/OpenIoT/opensensefeature"
fields="humidity,temperature"
field.temperature.propertyName="http://lsm.der.i.e/OpenIoT/Temperature"
field.temperature.unit=C
field.humidity.propertyName="http://lsm.der.i.e/OpenIoT/Humidity"
field.humidity.unit=Percent
field.co.propertyName="http://lsm.der.i.e/OpenIoT/CO"
field.co.unit=PPM
latitude=46.529838
longitude=6.596818
```

3.4.2.2.4.2.3 REGISTERING SENSORS TO LSM

Sensors can be registered to LSM by executing the script **lsm-register.sh** (on Linux/Mac) or **lsm-register.bat** (on Windows). This script takes as argument the metadata file name.

Syntax:

```
./lsm-register.sh <metadata_filename>
lsm-register.bat <metadata_filename>
```

Example:

```
./lsm-register.sh virtual-sensors/opensense1.metadata
lsm-register.bat virtual-sensors\opensense1.metadata
```

After calling the registration scripts, LSM answers to the requests and assigns a sensor ID, like in this example:

```
Sensor registered to LSM with ID:
http://lsm.derri.ie/resource/557635232734257
```

3.4.2.2.4.2.4 PUSHING DATA TO LSM

In order to push data to LSM, the LSMExporter processing class needs to be used. This class can be found in package (org/openiot/gsn/vsensor). LSMExporter loads metadata from the metadata file and uses it to push annotated data to LSM whenever new data arrives from connected wrappers.

3.4.2.2.4.2.5 REGISTERING SENSOR DATA THROUGH AN RDF FILE

Alternatively to a .properties file, you can provide an RDF metadata description for your virtual sensor. Currently we support an RDF in standard Turtle format (<http://www.w3.org/TR/turtle/>). For example the following RDF metadata:

```
@prefix : <http://sensordb.csiro.au/id/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix DUL: <http://www.loa-cnr.it/ontologies/DUL.owl#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix qu: <http://purl.oclc.org/NET/ssnx/qu/qu#> .
@prefix unit: <http://purl.oclc.org/NET/ssnx/qu/unit#> .
@prefix aws: <http://purl.oclc.org/NET/ssnx/meteo/aws#> .
@prefix cf-property: <http://purl.oclc.org/NET/ssnx/cf/cf-property#> .
@prefix cf-feature: <http://purl.oclc.org/NET/ssnx/cf/cf-feature#> .
@prefix prov: <http://purl.org/net/provenance/ns#> .
@prefix phenonet: <http://sensordb.csiro.au/ontology/phenonet#> .
@prefix wgs84: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix lgd: <http://linkedgeodata.org/property/> .
@prefix lsm: <http://lsm.derri.ie/ont/lsm.owl#> .
@base <http://sensordb.csiro.au/id/> .

<sensor/5010> rdf:type aws:CapacitiveBead,ssn:Sensor;
  rdfs:label "Sensor 5010";
  ssn:observes aws:air_temperature ;
  phenonet:hasSerialNumber <sensor/5010/serial/serial2> ;
  ssn:onPlatform <site/narrabri/Pweather> ;
  ssn:ofFeature <site/narrabri/sf/sf_narrabri> ;
  ssn:hasMeasurementProperty <sensor/5010/accuracy/acc_1> ;
  prov:PerformedBy "SensorSource";
  DUL:hasLocation <place/location1>;
  lsm:hasSensorType <sensorType1>;
  lsm:hasSourceType "SourceType".

<sensorType1> rdfs:label "TypeName".
<place/location1> rdf:type DUL:Place;
  wgs84:lat 52.3;
  wgs84:long 98.2;
  lsm:is_in_city <city1>;
  lgd:is_in_province <prov1>;
  lgd:is_in_country <country1>;
  lgd:is_in_continent <conti>.

<site/narrabri/Pweather> rdf:type ssn:Platform ;
  ssn:inDeployment <site/narrabri/deployment/2013> .
<site/narrabri/sf/sf_narrabri> rdf:type phenonet:SamplingFeature;
  ssn:ofFeature cf-feature:atmosphere_air .
aws:air_temperature qu:unit phenonet:degreeCelsius;
  rr:columnName "temperature" .
<city1> rdfs:label "cityname".
<prov1> rdfs:label "province".
<country1> rdfs:label "country".
<conti> rdfs:label "conti".
```

In order to execute the sensor registration you may proceed as follows (adding the `-rdf` parameter):

```
./lsm-register.sh virtual-sensors/opensense1.ttl -rdf
```

3.4.2.2.4.2.6 REGISTERING SENSOR DATA THROUGH A REST SERVICE

Instead of using the `lsm_register` command line tool, you can register the sensor metadata with the following POST request, passing the file contents as body of the request:

```
http://GSNSERVER:22001/vs/vsensor/VSENSOR_NAME/register
```

You can use any REST or HTTP library to execute this request in your favourite programming language. Or if you want to test it using the `curl` client:

```
curl -v http://localhost:22001/vs/vsensor/sens123/register -X POST --data-binary @virtual-sensors/LSM/opensense_1.metadata
```

There is also a service to post RDF data:

```
http://GSNSERVER:22001/vs/vsensor/VSENSOR_NAME/registerRdf
```

And a service to create a GSN virtual sensor (the typical GSN xml file that has the wrapper information):

```
http://GSNSERVER:22001/vs/vsensor/VSENSOR_NAME/create
```

This allows to remotely install a virtual sensor in X-GSN, and register the metadata in LSM.

3.5 Security Module

3.5.1 Main Released Functionalities & Services

The Security module consists of three sub-modules, namely, *security-server*, *security-client*, and *security-management*. In the following sections, the functionality of each sub-module is described.

3.5.1.1 Security-server

The security-server is based on Jasig CAS¹⁶ and provides OAuth2.0 authentication and authorization for all other OpenIoT modules. The security-server module provides the following functionality:

1. Form based single sign-on for web applications using OAuth2.0 protocol. A token with limited lifetime is returned to the web application after a successful login.
2. Restful API for programmatically logging in to CAS and obtain a token.
3. Permissions assigned to the user represented by the previously obtained token can be delivered to the requesting client. The permissions are used for controlling access of users to various OpenIoT resources.

3.5.1.2 Security-client

The security-client module provides the required API for the developers to access the security-server and perform access control. We distinguish between two types of OpenIoT applications: (I) web applications: that provide user interfaces that run in a browser and (II) REST applications: that run in JBoss and users interact with them indirectly through a web application. The security-client module provides the following functionality for each application type:

Web applications:

- Protecting URLs so that only the users with sufficient permissions can access them
- Automatic or direct redirection of the user to the login page of OpenIoT security-server for authentication and obtaining a token
- Access control utility methods
- JSF and JSP tag libraries for access control in JSF and JSP pages
- Token validity check

REST applications:

- API for logging in and out
- Access control utility methods
- Token validity check

¹⁶ <http://www.jasig.org/cas>

3.5.1.3 Security-management

The security-management is the management console of OpenIoT security module. It provides a user interface with the following functionality:

- Adding/managing services
- Defining/managing permissions for registered services
- Defining/managing roles for registered services
- Assigning roles to users or revoking roles from users
- Registering new users

In this context, the term **service** is any OpenIoT application or module that needs to use the security module for performing access control. For example, LSM-Light.Server, X-GSN, SD&UM, and Scheduler are referred to as services.

3.5.2 Install and Run

3.5.2.1 Developer

3.5.2.1.1 System requirements

All you need to build this project is Java 7.0 (Java SDK 1.7) or later, and Maven 3.0. Security-server and security-management applications are designed to be run on JBoss Enterprise Application Platform 6 or JBoss AS 7.1.

3.5.2.1.2 Download

To download the source code of security modules 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 security modules are available under the “openiot/modules/security/” folder.

3.5.2.1.3 Deploy and Run

Security-server depends on utils.common and lsm-light.client modules. So they must be available in the local Maven repository in order to be able to build the security server module. It also depends on lsm-light.server. Therefore, lsm-light.server must be already deployed in JBoss. For deploying the security-server module, run the following command in “openiot/modules/security/security-server” folder:

```
mvn jboss-as:deploy
```

The security-server module will be deployed in JBoss under the name openiot-cas and it is then accessible by default on <https://localhost:8443/openiot-cas>.

Security-client depends on utils.common. Therefore, it must be available in the local Maven repository in order to be able to build the security client module. In order to build and install the security-client module in the local maven repository, run the following command in “openiot/modules/security/security-client” folder:

```
mvn install
```

Security-management depends on utils.common, lsm-light.client and security-client modules. So they must be available in the local maven repository in order to be able to build the security server module. For deploying the security-management module run the following command in “openiot/modules/security/security-management” folder:

```
mvn jboss-as:deploy
```

If the security console is not deployed on localhost, before deploying it, copy “openiot/modules/security/security-management/src/main/resources/web-client.ini” to your JBoss configuration folder under the name “web-client-security.management.ini” and modify the address of the security-server. Security console is accessible on <http://localhost:8080/security.management> if it is deployed locally.

3.5.2.2 User

3.5.2.2.1 System requirements

All you need to run security modules is Java 7.0 (Java SDK 1.7) or later, and JBoss Enterprise Application Platform 6 or JBoss AS 7.1.

You can download the binaries through the OpenIoT Wiki ¹⁷ under the Users>Downloads¹⁸ section.

3.5.2.2.2 Deploy and Run

Put the openiot-cas.war and security.management.war into the deployment directory of your JBoss installation or use your JBoss administration console to deploy these two web applications. Note that openiot-cas.war requires that lsm-light.server be already installed. You can modify the properties pertinent to the OpenIoT security module in the global openiot.properties file, if the default settings do not comply with your installation.

¹⁷ <https://github.com/OpenIoTOrg/openiot/wiki>

¹⁸ <https://github.com/OpenIoTOrg/openiot/wiki/Downloads>

3.6 CUPUS

The CUPUS middleware consists of the Cloud Broker and external entities such as subscribers, publishers or mobile brokers. CUPUS provides an alternative data provision concept for better real time support. The OpenIoT Cloud Broker, i.e., the CPSP engine, accepts all messages from subscriber, publisher or mobile broker entities, based on user or ICO inputs. After processing the message, the cloud broker if necessary sends a notification or subscription to the end user entity (a subscriber, publisher or mobile broker). Hereinafter we describe how to set up and run only OpenIoT core component, i.e. the Cloud Broker.

3.6.1 Main Functionalities & Services

The current release of the OpenIoT Cloud Broker implements the functionalities/capabilities that are reflected in the Table 9.

Table 9: Cloud Broker's public methods and constructor

```
Broker (name:String, brokerPort: int): Broker
---
run (): void
shutdown(): void
```

Service descriptions as well as their inputs and outputs are listed in Table 10.

Table 10: Implemented Cloud Broker API definition

Service Name	Input	Output	Info
Broker (constructor)	String name, int brokerPort	Broker	Used to create a Broker entity. Requires as input an entity name in String format and port number. The output is a Broker object which is used for processing of all incoming messages on a specified port.
Run		void	A method used to start the cloud broker.
Shutdown		void	A method used to stop the cloud broker. All data (i.e. valid subscriptions and publications) remain in memory until the broker object is destroyed.

3.6.2 Install and Run

3.6.2.1 System requirements

All you need to build this project is Java 7.0 (Java SDK 1.7) or later, and Maven 3.0 or later. The service of this project is designed to be run on a cloud server.

3.6.2.2 Download

To download Cloud Broker'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 cloud broker is available under the “openiot/modules/CUPUS” folder

3.6.2.3 Deploy from the source code

If you have not yet done so, you must configure Maven before attempting the Cloud Broker deployment. After that:

- Build and deploy the Cloud Broker
 - NOTE: The following build command assumes that you have configured your Maven user settings. If you have not, you must include Maven setting arguments on the command line.
 - 1. Open a command line and navigate to the root directory of the CUPUS project.
 - 2. Type this command to build and deploy the archive:
 - `mvn package`
 - 3. This will build the service in target folder and now service is ready to be started by command in terminal: `java -jar target\StartBroker.jar <path_to_config_file19> <classpath>20`
- Undeploy the Cloud Broker service
 - 1. Stop the running instance of the service by typing “QUIT”

¹⁹ An example of the config file is given in the source/main/resource folder

²⁰ Classpath file is CUPUS-1.0.jar

3.6.2.4 Run in Eclipse

To integrate and deploy the Cloud Broker in Eclipse one should follow the steps below:

1. Import the existing Maven project “File>Import>Maven>Existing Maven Projects” - use the “Browse” button to navigate to the CUPUS source code directory that has been previously downloaded.
2. Click the Finish button.
3. Open the org.openiot.cupus.examples package
4. Right click on the “BrokerExample.java” choose “Run As>Java Application”

To undeploy the Cloud Broker follow the steps below:

1. Type “QUIT” or stop the system in the Console window

3.6.2.5 Run as a Service

Deploy: To deploy the Cloud Broker service, copy the “CloudBroker.java” to the cloud instance. Run the terminal command: `java -jar StartBroker.jar <path_to_config_file> <classpath>`.

Undeploy: Stop the running instance of the service by typing “QUIT”.

3.7 QoS Manager

The OpenIoT QoS Manager accepts all messages from the CUPUS middleware. After processing the messages, the QoS Manager entity administrates the data acquisition process from various data sources in the system, collects all readings, and publishes average sensor readings within the observed area. Additionally, the QoS manager is forwarding all sensor readings to the OpenIoT cloud database, by using X-GSN.

3.7.1 Main Functionalities & Services

The current release of the OpenIoT QoS Manager implements the functionalities/capabilities that are reflected in the interface listed in Table 11 while service descriptions as well as their inputs and outputs are listed in Table 12.

Table 11: List of primitives comprising the implemented QoS Manager

```
<<interface>>
QoSManager
---
setBatteryLevels (double highPriorityLevel, double lowPriorityLevel): void
setNumberOfActiveSensors (int numOfActiveSensors): void
getAllAvailableSensors (): Set<String>
getAllSensorsInArea (String area): Set<String>
getActiveSensorsInArea (String area): Set<String>
getAllSubscriptionsInArea (String area): List<TripletSubscription>
getAverageSensorReadingsInArea(String area): HashtablePublication
getLatLongFromArea (String area): List<Float>
getAreaFromLatLong (double lat, double lng, int accuracy): String
```

Table 12: Implemented QoS Manager API definition

Service Name	Input	Output	Info
setBatteryLevels	double highBatteryLevel, double lowBatteryLevel	void	A method used to set high and low battery level boundaries.
setNumberOfActive Sensors	int numberOfActive Sensors	void	A method used to set maximal number of active sensors in the observed area.

getAllAvailableSensors		Set<String>	Used to get all currently announced sensors in the system.
getAllSensorsInArea	String area	Set<String>	Used to get all sensors announced in the specific area.
getActiveSensorsInArea	String area	Set<String>	Used to get all active sensors in the specific area.
getAllSubscriptionsInArea	String area	List<TripletSubscription>	Used to get all end-user subscriptions in the specific area.
getAverageSensorReadingsInArea	String area	Hashtable Publication	A method used to get average sensor readings in the observed area.
getLatLongFromArea	String area	List<Float>	A method used to return latitude and longitude for the specific area.
getAreaFromLatLong	float lat, float lng, int accuracy	String	A method used to return area for the specific latitude and longitude.

3.7.2 Install and Run

3.7.2.1 System requirements

All that is necessary to build this project is Java 7.0 (Java SDK 1.7) or later, and Maven 3.0 or later, and the CUPUS library installed to your local Maven repository. The applications that this project produces are designed to be run on JBoss Enterprise Application Platform 6 or JBoss AS 7.1.

3.7.2.2 Download

To download QoS Manager'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 QoS Manager is available under the "openiot/modules/QoSManager" folder.

3.7.3 User

3.7.3.1 System requirements

All you need to run this project is Java 7.0 (Java SDK 1.7) or later, and JBoss Enterprise Application Platform 6 or JBoss AS 7.1, and the CUPUS middleware.

You can download the binaries through the OpenIoT Wiki ²¹ under the Users>Downloads²² section.

3.7.3.2 Published Interface

In case you would like to use a third party application to invoke QoS Manager services, use SOAP web services at the URLs listed below (the inputs and outputs of the services are described in Table 4 above):

- Welcome message listing the available services:
 - <http://localhost:8080/QoSManager/>
- Set battery level thresholds :
 - <http://localhost:8080/QoSManager/setBatteryLevels>
- Set number of active sensors per area :
 - <http://localhost:8080/QoSManager/setNumberOfActiveSensors>
- Get all currently available sensors in the system :
 - <http://localhost:8080/QoSManager/getAllSensors>
- Get all sensors in specific area :
 - <http://localhost:8080/QoSManager/getAllSensorsInArea>
- Get all active sensors in specific area :
 - <http://localhost:8080/QoSManager/getActiveSensorsInArea>
- Get all subscriptions in specific area :
 - <http://localhost:8080/QoSManager/getSubscriptionsInArea>
- Get average sensor data readings in specific area :
 - <http://localhost:8080/QoSManager/average>
- Get latitude and longitude points from specific MGRS area :
 - <http://localhost:8080/QoSManager/getLatLongFromArea>
- Get MGRS area from latitude and longitude points :
 - <http://localhost:8080/QoSManager/getAreaFromLatLng>

²¹ <https://github.com/OpenIoTOrg/openiot/wiki>

²² <https://github.com/OpenIoTOrg/openiot/wiki/Downloads>

3.8 User Interfaces

3.8.1 Request Definition

The request definition module 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” (OAMOs). Each application includes a collection of different services (OSMOs) which represent real life data (i.e. weather reports). This enables end-users to manage (describe/register/edit/update) their applications from a single user interface.

All modelled services are stored by the OpenIoT Scheduler and are automatically loaded when a user accesses the web application. More details and a user manual for this tool can be found in deliverable D4.4.2 “OpenIoT Integrated Development Environment”, and an example of its usage can be found in chapter 7 “OpenIoT End to End Demonstration”.

3.8.1.1 Main Functionalities & Services

Figure 4 illustrates the main application interface components:

- The 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 at the top right of the menu bar.
- The central pane serves as the workspace area for modelling services.
- The node toolbox (left pane) contains the list of nodes that can be dragged into the workspace. Nodes are grouped by functionality.
- The properties pane (right pane) provides access to any selected node's properties.
- The 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.

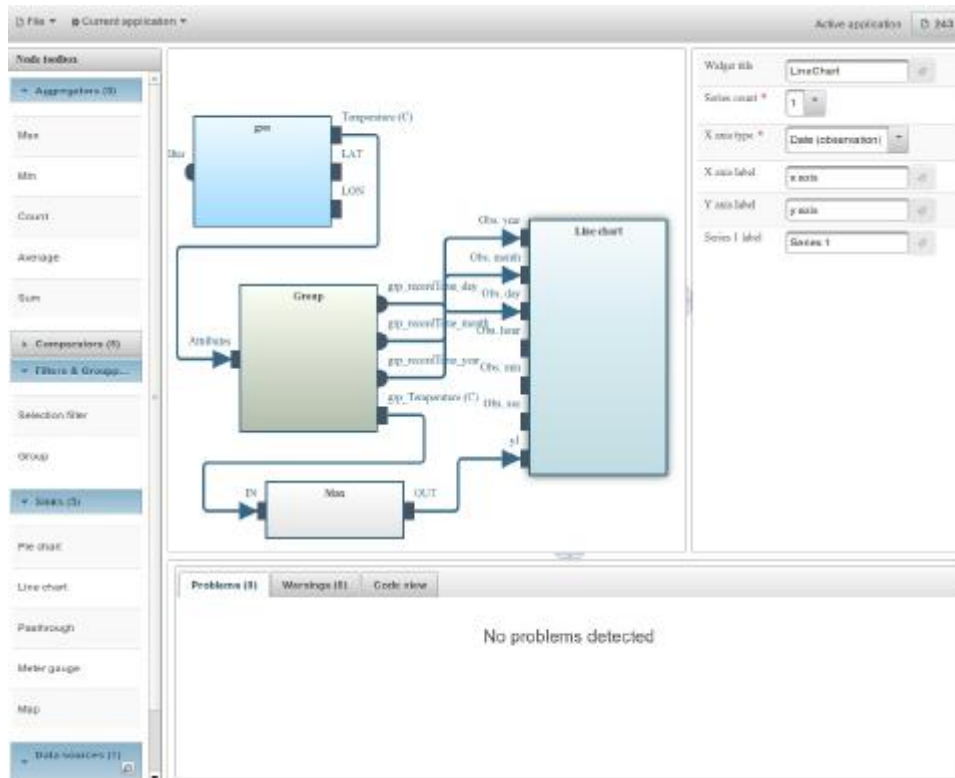


Figure 4: Request Definition User Interface (UI)

Below is a list of the Request Presentation main functionalities:

- Create a new application
- Load and Edit an existing application
- Modelling the service graph of an application by the configuration and usage of the provided toolboxes and more specifically the:
 - Data source node
 - Selection filter node
 - Comparator nodes
 - Group node
 - Aggregation nodes
 - Sink nodes
 - Line chart sink node
 - Pie chart sink node
 - Meter gauge sink node
 - Map sink node
 - Passthrough sink node
- Workspace validation
- Save/update an application

3.8.1.2 Install and Run

3.8.1.2.1 Developer

3.8.1.2.1.1 SYSTEM REQUIREMENTS

All you need to build this project is:

- Java 7.0 (Java SDK 1.7) or later,
- Maven 3.0 or later
- ui.requestCommons library (openiot/ui/ui.requestCommons) installed to your local maven repository,
- utils.commons library (openiot/utils/utils.commons/) installed to your local maven repository, and
- the Scheduler (see Section 3.1.2).

The application this project produces is designed to be run on JBoss Enterprise Application Platform 6 or JBoss AS 7.1.

3.8.1.2.1.2 DOWNLOAD

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

- HTTPS: <https://github.com/OpenlotOrg/openiot.git>
- SSH: <git@github.com:OpenlotOrg/openiot.git>

The Request Definition is available under the "openiot/ui/ui.requestDefinition/" folder. Directions on how to build and run the project from the source code can be found in chapter 5 "System Instalation for Use and Development"

Accessing the application:

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

3.8.1.2.1.3 DEFINING NEW NODES

With the exception of the sensor nodes which are populated dynamically from a sensor discovery query, all other nodes are specially annotated POJOs (Plain Old Java Objects) 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, then 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

3.8.1.2.1.4 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*.

3.8.1.2.2 User

3.8.1.2.2.1 SYSTEM REQUIREMENTS

To run this project you require Java 7.0 (Java SDK 1.7) or later, and JBoss Enterprise Application Platform 6 or JBoss AS 7.1.

You can download the binaries through the OpenIoT Wiki²³ under the Users>Downloads²⁴ section.

Directions on how to deploy the project from its binaries can be found in chapter 5 “System Instalation for Use and Development”

Accessing the application:

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

²³ <https://github.com/OpenIoTOrg/openiot/wiki>

²⁴ <https://github.com/OpenIoTOrg/openiot/wiki/Downloads>

3.8.2 Request Presentation

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.

To access the widget dashboard for a particular application, click on the file menu and then the open application sub-menu, and select an application to load. The request presentation layer will parse the application metadata and generate a self-updating widget dashboard (Figure 5).

Dashboards refresh automatically every 30 seconds. However, the user may manually trigger an update by clicking on the current application menu and selecting the “Manual data refresh” option. To clear the data of a specific widget, click on the “Clear data” button on its top-right corner.

More details and a user manual for this tool can be found in deliverable D4.4.2 “OpenIoT Integrated Development Environment” and a complete example of its usage can be found in chapter 7 “OpenIoT End to End Demonstration”.

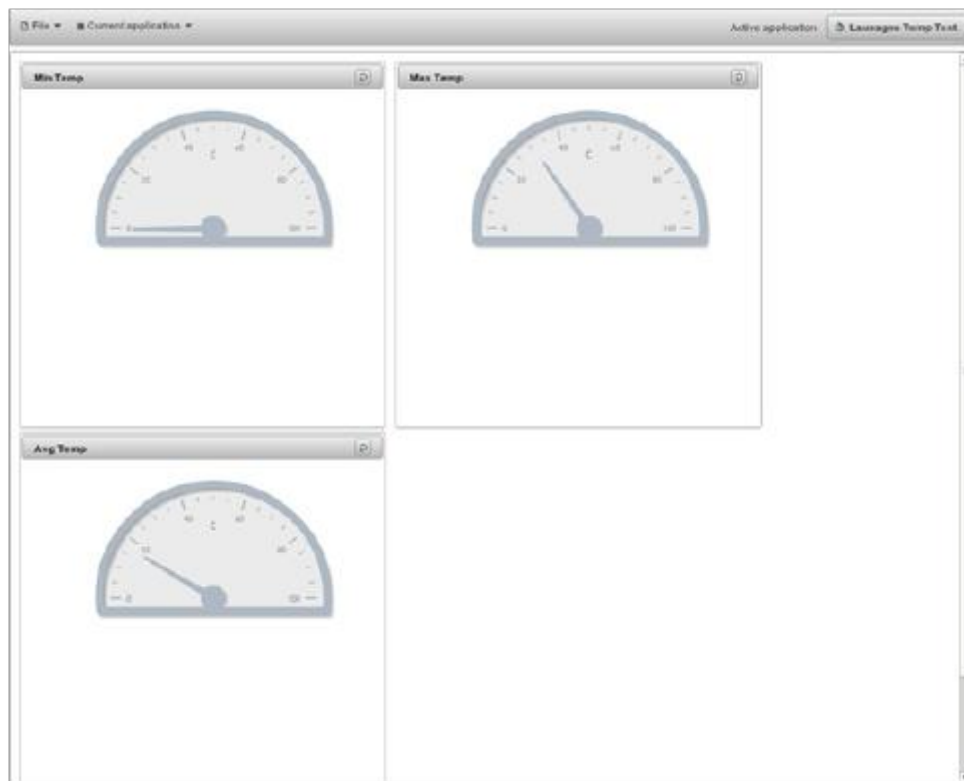


Figure 5: Request Presentation UI

3.8.2.1 Install and Run

3.8.2.1.1 Developer

3.8.2.1.1.1 SYSTEM REQUIREMENTS

All you need to build this project is:

- Java 7.0 (Java SDK 1.7) or later,
- Maven 3.0 or later
- ui.requestCommons library (openiot/ui/ui.requestCommons) installed to your local maven repository,
- utils.common library (openiot/utils/utils.common/) installed to your local maven repository, and
- the SD&UM (see Section 3.2).

The application this project produces is designed to be run on JBoss Enterprise Application Platform 6 or JBoss AS 7.1.

3.8.2.1.1.2 DOWNLOAD

To download Request Presentation'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 is available under the "openiot/ui/ui.requestPresentation/" folder.

Directions on how to build and run the project from the source code can be found in chapter 5 "System Installation for Use and Development"

Accessing the application:

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

3.8.2.1.1.3 ADDING A NEW SINK NODE WIDGET RENDERER

All widget implementations should be placed under the *org.openiot.ui.request.presentation.web.model.nodes.impl* package and should implement the *VisualizationWidget* interface. This interface defines three methods:

- *createWidget*. This method receives as input the list of presentation attributes that were filled in by the user during the service design and returns a JSF component instance that handles the widget view. The implementation is responsible for generating the widget wrapper (dashboard panel) and the actual widget renderer.
- *processData*. This method is invoked when new data is available for processing. The method receives a *SdumSeviceResultSet* as its input. The implementation is responsible for parsing the result set and updating its view accordingly.

- *clearData*. This method is invoked when the user clicks the widget's clear data button. The implementation is responsible for cleaning up any collected data and updating its view.

3.8.2.1.1.4 LOCALIZATION SUPPORT

The request presentation application has full i18n localization support via property files. These files are placed under the *org.openiot.ui.request.presentation.web.i18n* package.

3.8.2.1.2 User

3.8.2.1.2.1 SYSTEM REQUIREMENTS

To run this project you require Java 7.0 (Java SDK 1.7) or later, and JBoss Enterprise Application Platform 6 or JBoss AS 7.1.

You can download the binaries through the OpenIoT Wiki ²⁵ under the Users>Downloads²⁶ section.

Directions on how to deploy the project from its binaries can be found in chapter 5 “System Instalation for Use and Development”

Accessing the application:

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

²⁵ <https://github.com/OpenIoTOrg/openiot/wiki>

²⁶ <https://github.com/OpenIoTOrg/openiot/wiki/Downloads>

3.8.3 Schema Editor

3.8.3.1 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 and the backend LD4Sensors Server. The Schema Editor interface depends on the LD4Sensors Server for generating descriptions of sensor metadata and sensor observations.

The LD4Sensor exposes a REST API that takes as input the sensor metadata 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/>).

3.8.3.2 Download

The sensor schema editor can be downloaded from <https://github.com/OpenIoTOrg/openiot/tree/develop/ui/RDFSensorSchemaEditor>

3.8.3.3 Deploy

Prerequisite

LD4Sensors 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.

Deployment/Undeployment

The Sensor Schema Editor is deployed in a JBoss AS 7.1.X container. The following instructions provide the steps to deploy and undeploy the RDFSchema Editor.

Deploy: To deploy the RDF Schema Editor on JBoss AS 7.X, 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.

Directions on how to build and run the project from the source code can be found in chapter 5 “System Instalation for Use and Development”

3.8.3.4 Use

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.

3.9 Utilities & Libraries

3.9.1 Commons library

Finally in OpenIoT a project called Commons is maintained where the "common" Objects, Schemata and utilities used for most of the modules across the OpenIoT platform are stored and developed. This project is included in the binary file distribution as a library to all the projects that are using it but has to be added as a separate project for development. This project is available under the `/utils/Utils.Commons/`²⁷ folder.

The modules that are currently using this library are the:

- Request Definition
- Request Presentation
- Service Delivery & Utility Manager (SD&UM)
- Scheduler
- LSM-Light

Currently `util.commons` hosts the following objects, schemata, utilities:

- Schemata (xsd):
 - OSDSpec
 - SdumServiceResultSet
 - SPARQL
 - protocol-types,
 - rdf,
 - result
 - AppUsageReport
 - DescriptiveIDs
 - SensorTypes
- Java models generated from schemata:
 - descriptiveids
 - osdspec
 - SDUM
 - appusagereport
 - serviceresultset
 - sensortypes
 - sparql
 - protocol-types,
 - rdf,
 - result
- Utilities
 - CDataAdapter
 - PropertyManagement

²⁷ <https://github.com/OpenIoTOrg/openiot/tree/master/utils/Utils.Commons>

3.9.1.1 Download

To download Commons' 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 Commons is available under the “openiot /utils /utils.common/” folder.

As this project is a maven dependency it has to be installed to the local maven repository before starting the developments on one of the aforementioned modules. If you have not yet done so, you must install Maven.

Directions on how to build and install the project from the source code can be found in chapter 5 “System Instalation for Use and Development”

4 SOURCE CODE & BINARIES

4.1 Source Code Availability

The OpenIoT repository is hosted at the GitHub²⁸ and can be found at the following link: <https://github.com/OpenIoTOrg/openiot>

The OpenIoT repository is divided into branches. Each branch is separated into two thematic categories. One is the Documentation (i.e., site storage hosted at the “gh-pages”). And the other one is the Open IoT source code branches. Under the source code category various Branches will exist that are listed below:

- Main branches with an infinite lifetime:
 - Master branch
 - Develop branch
- Supporting branches:
 - Feature branches
 - Release branches
 - Hotfix branches

4.2 Source code Structure

The OpenIoT source code is organised in functionality themes. For example all utilities are under the “utils” folder and all user interfaces under the “ui” folder. The code structure is shown below:

- doc: provides all the related documents with the platform.
- Modules: provides the core modules of the platform
 - CUPUS
 - QoSManager
 - x-gsn
 - scheduler
 - scheduler.core
 - scheduler.client
 - sdum
 - sdum.core
 - sdum.client
 - lsm-light
 - lsm-light.client
 - lsm-light.server
 - security
 - security-client
 - security-server
 - security-webapp-demo

²⁸ <http://github.com/>

- sandbox: provides space for developers to store their test code/apps (developers “playground”).
- Ui: provides all the modules related to the User Interface
 - RDFSensorSchemaEditor
 - Ide.core
 - ui.requestCommonsminor
 - ui.requestDefinition
 - ui.requestPresentationaEditor
- utils: provides utilities related with the platform
 - demoData
 - lib
 - utils.common

4.3 Binaries Availability

OpenIoT binaries are available through the OpenIoT Wiki site²⁹ under the Users>Downloads³⁰ section. They follow the versioning of the stable releases and are currently in the alpha stage. They are available as standalone executables per module that can be downloaded separately or in groups where one can download the complete platform in one zip file. Moreover an option to download a complete preinstalled and functional VirtualBox³¹ is provided for download to help the adoption and testing of the platform. The usage and specifications of the VirtualBox is described in the following section.

4.4 Complete Package as a Virtual Machine

The OpenIoT release v0.1.1 is available as a virtual box image. The recommended Configuration for VirtualBox (tested with the below configuration for stable operation) is:

- 2 GB Memory
- 12 GB HDD (6 GB Linux, 3.5 GB Data expandable to 5.5GB using GParted tool for Ubuntu, 2 GB Swap)
- Two core processing

The OpenIoT release v0.1.1 has been deployed on Ubuntu 12.04. The virtual box image has been pre-configured with all the pre-requisites for testing and development purposes.

4.4.1 Deploying and Starting VM

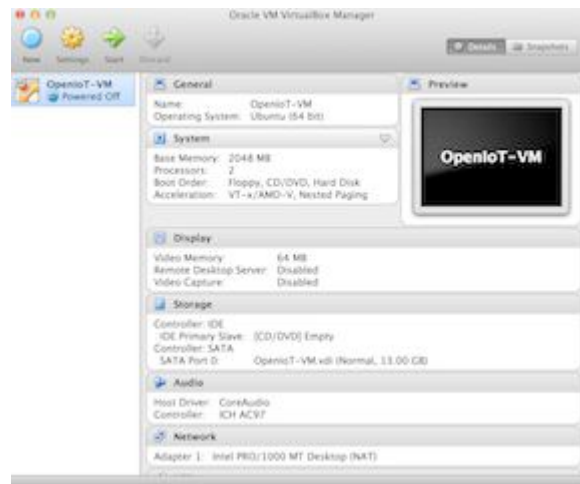
- Install the Oracle Virtual Box

²⁹ <https://github.com/OpenIoTOrg/openiot/wiki>

³⁰ <https://github.com/OpenIoTOrg/openiot/wiki/Downloads>

³¹ <https://www.virtualbox.org/>

- From the location where the VM is stored, double click the file OpenIoT-VM.vbox. A window as shown below will open with the OpenIoT-VM settings.
- Click the Start icon to start the VM.



4.4.2 Running OpenIoT Services

4.4.2.1 Common Setup Configuration

- The username and password for the VM is located in an instruction.txt file in the downloaded VM Image package.
- The scripts required to run the OpenIoT services namely Scheduler, SDUM, Request Definition, Request Presentation, LSM Server, X-GSN and Virtuoso are located in the Desktop under the *scripts* directory

4.4.2.2 OpenIoT Property File

An `openiot.properties` file located in the JBOSS configuration folder `/media/openiot/OpenIoT-Dev-Environment/jboss-as-7.1.1.Final/standalone/configuration` provides the configuration information to OpenIoT services.

- The OpenIoT VM determined by the `openiot.properties` can be deployed in the following run-time configurations
 - Run all components locally
 - Use an existing installation of virtuoso and LSM-Server running on DERI's infrastructure

Note 1: By using DERI's infrastructure, the OpenIoT Platform will have access to public data pushed by other OpenIoT users to DERI's server.

Note 2: A local installation is suitable for users who intend to use OpenIoT services within the organisation's network providing complete control over sharing of data to external users.

4.4.2.3 Local Deployment Property File Example

```
#Scheduler Properties
scheduler.core.lsm.openiotMetaGraph=http://openiot.eu/OpenIoT/sensormeta#
scheduler.core.lsm.openiotDataGraph=http://openiot.eu/OpenIoT/sensordata#
scheduler.core.lsm.openiotFunctionalGraph=http://openiot.eu/OpenIoT/functionaldata#
scheduler.core.lsm.access.username=openiot_guest
scheduler.core.lsm.access.password=openiot
scheduler.core.lsm.sparql.endpoint=http://localhost:8890/sparql
scheduler.core.lsm.remote.server=http://localhost:8080/lsm-light.server/
#Service Delivery & Utility Manager (SD&UM) Properties
sdum.core.lsm.openiotFunctionalGraph=http://openiot.eu/OpenIoT/functionaldata#
sdum.core.lsm.sparql.endpoint=http://localhost:8890/sparql
sdum.core.lsm.remote.server=http://localhost:8080/lsm-light.server/
#Request Definition
#Request Presentation
#LSM-LIGHT Properties
lsm-light.server.connection.driver_class=virtuoso.jdbc4.Driver
lsm-light.server.connection.url=jdbc:virtuoso://localhost:1111/log_enable=2
lsm-light.server.connection.username=dba
lsm-light.server.connection.password=dba
lsm-light.server.minConnection=10
lsm-light.server.maxConnection=15
lsm-light.server.acquireRetryAttempts=5
#for local virtuoso instance
lsm-light.server.localMetaGraph = http://openiot.eu/OpenIoT/sensormeta#
lsm-light.server.localDataGraph = http://openiot.eu/OpenIoT/sensordata#
```

4.4.2.4 DERI's Infrastructure-based Deployment Property File

```
#Scheduler Properties
scheduler.core.lsm.openiotMetaGraph=http://lsm.deri.ie/OpenIoT/guest/sensormeta#
scheduler.core.lsm.openiotDataGraph=http://lsm.deri.ie/OpenIoT/guest/sensordata#
scheduler.core.lsm.openiotFunctionalGraph=http://lsm.deri.ie/OpenIoT/testSchema#
scheduler.core.lsm.access.username=openiot_guest
scheduler.core.lsm.access.password=openiot
scheduler.core.lsm.sparql.endpoint=http://lsm.deri.ie/sparql
scheduler.core.lsm.remote.server=http://lsm.deri.ie/lsm-light.server/

#Service Delivery & Utility Manager (SD&UM) Properties
```

```
s dum.core.lsm.openiotFunctionalGraph=http://lsm.der i.ie/OpenIoT/testSchema#
s dum.core.lsm.sparql.endpoint=http://lsm.der i.ie/sparql
s dum.core.lsm.remote.server=http://lsm.der i.ie/lsm-light.server/

#Request Definition
#Request Presentation
#LSM-LIGHT Properties
lsm-light.server.connection.driver_class=virtuoso.jdbc4.Driver
lsm-light.server.connection.url=jdbc:virtuoso://lsm.der i.ie:1111/log_enable=2
lsm-light.server.connection.username=<database username>
lsm-light.server.connection.password=<database password>
lsm-light.server.minConnection=10
lsm-light.server.maxConnection=15
lsm-light.server.acquireRetryAttempts=5

#for local virtuoso instance
lsm-light.server.localMetaGraph = http://lsm.der i.ie/OpenIoT/guest/sensormeta#
lsm-light.server.localDataGraph = http://lsm.der i.ie/OpenIoT/guest/sensordata#
```

4.4.3 OpenIoT's VirtualBox End User Manual

For an end-user who would like to use the platform as-is, perform the following steps

1. Make a choice for a local deployment or DERI infrastructure based deployment
2. Based on step 1 choice, navigate to the folder /media/openiot/OpenIoT-Dev-Environment/jboss-as-7.1.1.Final/standalone/configuration
 - a. Rename the openiot.properties.local file to openiot.properties if you choose a local setup
 - b. Rename the openiot.properties.der i file to openiot.properties if you choose a DERI's based setup
3. Open a terminal and navigate to /home/openiot/Desktop/scripts
4. Starting Virtuoso
 - a. Run the script ./virtuoso_start
 - b. You will see the output as below


```

openiot@openiot-VirtualBox: ~/Desktop/scripts
14:10:55 FAILED plugin 2: Unable to locate file }
14:10:55 { Loading plugin 3: Type 'plain', file 'creolewiki' in '/usr/local/virtuoso-opensource/lib/virtuoso/hosting'
14:10:55 FAILED plugin 3: Unable to locate file }
14:10:55 OpenLink Virtuoso Universal Server
14:10:55 Version 07.00.3203-pthreads for Linux as of Dec 11 2013
14:10:55 uses parts of OpenSSL, PCRE, Httl Tidy
14:10:55 Database version 3126
14:10:55 SQL Optimizer enabled (max 1000 layouts)

openiot@openiot-VirtualBox:~/Desktop/scripts$ 14:10:56 Compiler unit is timed at
0.000181 msec
14:10:57 built-in procedure "repl_undot_name" overruled by the RDBMS
14:10:57 Roll forward started
14:10:57 Roll forward complete
14:10:58 Checkpoint started
14:10:58 Checkpoint finished, log reused
14:10:58 HTTP/WebDAV server online at 8890
14:10:58 Server online at 1111 (pid 2274)

openiot@openiot-VirtualBox:~/Desktop/scripts$
openiot@openiot-VirtualBox:~/Desktop/scripts$
openiot@openiot-VirtualBox:~/Desktop/scripts$
openiot@openiot-VirtualBox:~/Desktop/scripts$

```

5. Starting JBOSS

- Run the script `./jboss_start`
- You will see the output as below

```

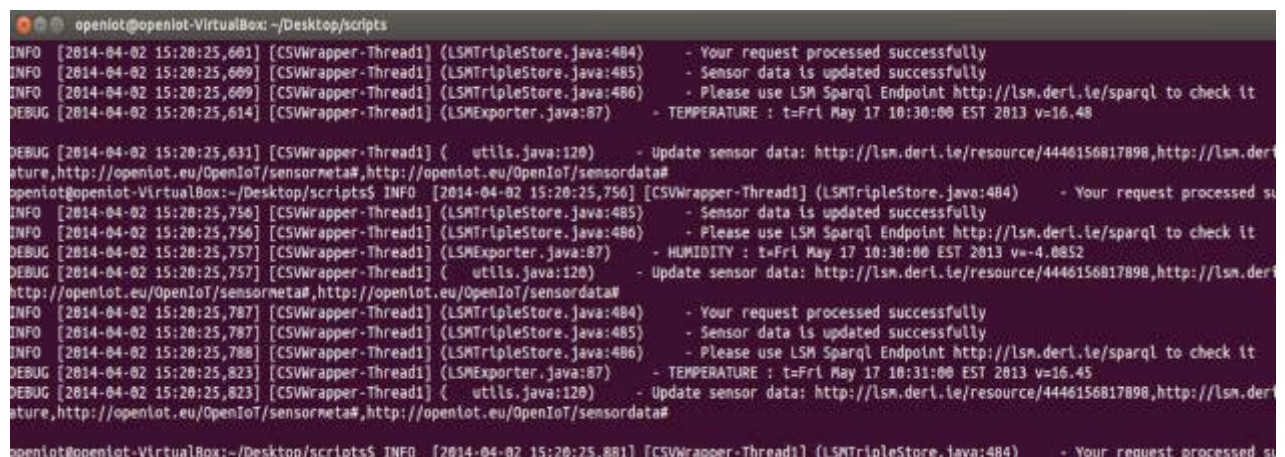
openiot@openiot-VirtualBox: ~/Desktop/scripts
S015893: Encountered invalid class name 'http://java.sun.com/jaxp/xpath/dom:
net.sf.saxon.xpath.XPathFactoryImpl' for service type 'javax.xml.xpath.XPathFac
tory'
14:24:36,529 WARN [org.jboss.as.server.deployment] (MSC service thread 1-3) JBA
S015893: Encountered invalid class name 'http://saxon.sf.net/jaxp/xpath/on:
net.sf.saxon.xpath.XPathFactoryImpl' for service type 'javax.xml.xpath.XPathFac
tory'
14:24:36,759 INFO [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
host].[/lsn-light.server]] (MSC service thread 1-4) No Spring WebApplicationInit
ializer types detected on classpath
14:24:36,766 INFO [org.openiot.lsm.pooling.ConnectionManager] (MSC service thre
ad 1-4) Loading property file
14:24:36,766 INFO [org.openiot.lsm.pooling.ConnectionManager] (MSC service thre
ad 1-4) loading database server driver virtuoso.jdbc4.Driver
14:24:37,049 INFO [org.openiot.lsm.pooling.ConnectionManager] (MSC service thre
ad 1-4) contextInitialized.....Connection Pooling is configured
14:24:37,052 INFO [org.openiot.lsm.pooling.ConnectionManager] (MSC service thre
ad 1-4) Total connections ==> 20
14:24:37,086 INFO [javax.enterprise.resource.webcontainer.jsf.config] (MSC serv
ice thread 1-4) Initializing Mojarra 2.1.7-jbossorg-1 (20120227-1401) for contex
t '/lsn-light.server'
14:24:38,153 INFO [org.jboss.web] (MSC service thread 1-4) JBAS018210: Register
ing web context: /lsn-light.server
14:24:38,165 INFO [org.jboss.as.server] (Controller Boot Thread) JBAS018559: De
ployed "lsn-light.server.war"
14:24:38,165 INFO [org.jboss.as.server] (Controller Boot Thread) JBAS018559: De
ployed "ui.requestPresentation.war"
14:24:38,166 INFO [org.jboss.as.server] (Controller Boot Thread) JBAS018559: De
ployed "ui.requestDefinition.war"
14:24:38,166 INFO [org.jboss.as.server] (Controller Boot Thread) JBAS018559: De
ployed "sdum.core.war"
14:24:38,166 INFO [org.jboss.as.server] (Controller Boot Thread) JBAS018559: De
ployed "scheduler.core.war"
14:24:38,183 INFO [org.jboss.as] (Controller Boot Thread) JBAS015951: Admin con
sole listening on http://127.0.0.1:9990
14:24:38,183 INFO [org.jboss.as] (Controller Boot Thread) JBAS015874: JBoss AS
7.1.1.Final "Brontes" started in 14505ms - Started 440 of 525 services (84 servi
ces are passive or on-demand)

```

- Note: If you start JBOSS without performing step 2, JBOSS will throw exceptions.

6. Starting X-GSN to push data into the OpenIoT system.

- A virtual sensor is used to configure X-GSN with appropriate data sources. Please refer to the wiki page for more details
- Depending on your choice in step 2, navigate to the location `/media/openiot/OpenIoT-Dev-Environment/develop-code-mar2014/openiot/modules/x-gsn/virtual-sensors`
 - There are two folders namely `local-install` and `deri-install`
 - Copy the files in from the directory that corresponds to your setup (Step 2) to the main `virtual-sensors` directory
 - These files contain the virtual sensor configuration and the corresponding metadata that describe the sensor. The sample file description are
 - Local instance: A weather station sensor located in Canberra, Australia. The data for the sensor is read from a CSV file stored in X-GSN folder under `data`
 - DERI instance: A weather station sensor located in Lausanne, Switzerland. The data for this sensor is read from a CSV file stored in the X-GSN folder under `data`
- Open a new terminal instance
- Go to `/home/openiot/Desktop/scripts`
- Run the script `./gsn_start.sh`
- You will see the output as below



```

openiot@openiot-VirtualBox: ~/Desktop/scripts
INFO [2014-04-02 15:20:25,601] [CSVWrapper-Thread1] (LSMTriplesStore.java:484) - Your request processed successfully
INFO [2014-04-02 15:20:25,609] [CSVWrapper-Thread1] (LSMTriplesStore.java:485) - Sensor data is updated successfully
INFO [2014-04-02 15:20:25,609] [CSVWrapper-Thread1] (LSMTriplesStore.java:486) - Please use LSM Sparql Endpoint http://lsm.deri.ie/sparql to check it
DEBUG [2014-04-02 15:20:25,614] [CSVWrapper-Thread1] (LSMExporter.java:87) - TEMPERATURE : t=Fri May 17 10:30:00 EST 2013 v=16.48
DEBUG [2014-04-02 15:20:25,631] [CSVWrapper-Thread1] (utils.java:120) - Update sensor data: http://lsm.deri.ie/resource/4446156817898,http://lsm.deri
ature,http://openiot.eu/OpenIoT/sensormeta#,http://openiot.eu/OpenIoT/sensordata#
openiot@openiot-VirtualBox:~/Desktop/scripts$ INFO [2014-04-02 15:20:25,756] [CSVWrapper-Thread1] (LSMTriplesStore.java:484) - Your request processed su
INFO [2014-04-02 15:20:25,756] [CSVWrapper-Thread1] (LSMTriplesStore.java:485) - Sensor data is updated successfully
INFO [2014-04-02 15:20:25,756] [CSVWrapper-Thread1] (LSMTriplesStore.java:486) - Please use LSM Sparql Endpoint http://lsm.deri.ie/sparql to check it
DEBUG [2014-04-02 15:20:25,757] [CSVWrapper-Thread1] (LSMExporter.java:87) - HUMIDITY : t=Fri May 17 10:30:00 EST 2013 v=-4.0852
DEBUG [2014-04-02 15:20:25,757] [CSVWrapper-Thread1] (utils.java:120) - Update sensor data: http://lsm.deri.ie/resource/4446156817898,http://lsm.deri
http://openiot.eu/OpenIoT/sensormeta#,http://openiot.eu/OpenIoT/sensordata#
INFO [2014-04-02 15:20:25,787] [CSVWrapper-Thread1] (LSMTriplesStore.java:484) - Your request processed successfully
INFO [2014-04-02 15:20:25,787] [CSVWrapper-Thread1] (LSMTriplesStore.java:485) - Sensor data is updated successfully
INFO [2014-04-02 15:20:25,788] [CSVWrapper-Thread1] (LSMTriplesStore.java:486) - Please use LSM Sparql Endpoint http://lsm.deri.ie/sparql to check it
DEBUG [2014-04-02 15:20:25,823] [CSVWrapper-Thread1] (LSMExporter.java:87) - TEMPERATURE : t=Fri May 17 10:31:00 EST 2013 v=16.45
DEBUG [2014-04-02 15:20:25,823] [CSVWrapper-Thread1] (utils.java:120) - Update sensor data: http://lsm.deri.ie/resource/4446156817898,http://lsm.deri
ature,http://openiot.eu/OpenIoT/sensormeta#,http://openiot.eu/OpenIoT/sensordata#
openiot@openiot-VirtualBox:~/Desktop/scripts$ INFO [2014-04-02 15:20:25,881] [CSVWrapper-Thread1] (LSMTriplesStore.java:484) - Your request processed su

```

- Note: If you run step 6 without completing the previous steps, you will see exceptions and errors. Please ensure the previous steps are completed without any errors.

7. Stop all services

- a. To stop all the services, please run the corresponding stop scripts in the following order
- b. `./gsn_stop.sh`
- c. `./jboss_stop.sh`
- d. `./virtuoso_stop.sh`

4.4.4 OpenIoT's VirtualBox Developer Manual

- The VM provides developers with ready access to an Eclipse environment with pre-configured eclipse plugins to support OpenIoT development.
- The user manual can be used to test the system out of the box
- The VM comes installed with git, maven, JBOSS, Virtuoso, JBOSS and maven plugins for eclipse allowing developer to update their code as and when new release of code is available
- The eclipse environment can be started from the desktop or from `/media/openiot/OpenIoT-Dev-Environment/`
- The entire OpenIoT dependencies including JBOSS and latest git code is available from `/media/openiot/OpenIoT-Dev-Environment/`

5 SYSTEM INSTALATION FOR USE AND DEVELOPMENT

5.1 Utilities & Properties

5.1.1 “Global” Properties

OpenIoT to ease its deployment and configuration is using a “global” properties file (openiot.properties³²), for the modules deployed within JBoss, which contains all the attributes that can be configured from the User. The modules that are currently configured to use this configuration file are the:

- Scheduler
- Service Delivery & Utility Manager
- Data Platform (LSM)
- QoS Manager
- IDE Core

The property file can be found under `utils/utils.commons/src/main/resources/properties/`. This file SHOULD be placed under the following path: `\JBoss_HOME\standalone\configuration` of your JBoss 7.XX container. The attributes that can be modified by using this property file can be identified in the following configuration example:

```
# IDE Core Navigation Properties

# Request Definition
ide.core.navigation.requestDefinition.title=Request Definition
ide.core.navigation.requestDefinition.url=http://localhost:8080/ui.requestDefinition/pages/applicationDesign.xhtml
ide.core.navigation.requestDefinition.monitoring=true

#Request Presentation
ide.core.navigation.requestPresentation.title=Request Presentation
ide.core.navigation.requestPresentation.url=http://localhost:8080/ui.requestPresentation/pages/requestPresentation.xhtml
ide.core.navigation.requestPresentation.monitoring=true

#Sensor Schema Editor
ide.core.navigation.sensorSchemaEditor.title=Sensor Schema Editor
ide.core.navigation.sensorSchemaEditor.url=http://localhost:8080/sensorschema/index.xhtml
ide.core.navigation.sensorSchemaEditor.monitoring=true

#Scheduler Properties

scheduler.core.lsm.openiotMetaGraph=http://lsm.der.i.e/OpenIoT/guest/sensormeta#
```

³²

<https://github.com/OpenlotOrg/openiot/tree/master/utils/utils.commons/src/main/resources/properties>

```

scheduler.core.lsm.openiotDataGraph=http://lsm.der.i.e/OpenIoT/guest/sensor
data#
scheduler.core.lsm.openiotFunctionalGraph=http://lsm.der.i.e/OpenIoT/guest/
functional#
scheduler.core.lsm.access.username=openiot_guest
scheduler.core.lsm.access.password=openiot
scheduler.core.lsm.sparql.endpoint=http://lsm.der.i.e/sparql
scheduler.core.lsm.remote.server=http://lsm.der.i.e/lsm-light.server/

#Service Delivery & Utility Manager (SD&UM) Properties

sdum.core.lsm.openiotFunctionalGraph=http://lsm.der.i.e/OpenIoT/guest/funct
ional#
sdum.core.lsm.sparql.endpoint=http://lsm.der.i.e/sparql
sdum.core.lsm.remote.server=http://lsm.der.i.e/lsm-light.server/

#LSM-LIGHT Properties

lsm-light.server.connection.driver_class=virtuoso.jdbc4.Driver
lsm-
light.server.connection.url=jdbc:virtuoso://lsm.der.i.e:1111/log_enable=2
lsm-light.server.connection.username=<database username>
lsm-light.server.connection.password=<database password>
lsm-light.server.minConnection=10
lsm-light.server.maxConnection=15
lsm-light.server.acquireRetryAttempts=5
#for local virtuoso instance
lsm-light.server.localMetaGraph = http://test/sensormeta#
lsm-light.server.localDataGraph = http://test/sensordta#

#QoS Manager

qos.name=QualityOfService
qos.brokerIP=localhost
qos.brokerPort=10000
qos.gsnAddress=localhost:22001
qos.wrapperPort=30000
qos.numberOfSensors = 3
qos.highBatteryLevel = 70
qos.lowBatteryLevel = 30
qos.sensorParameters =
Temperature, Humidity, Pressure, NO2, SO2, CO, BatteryS, BatteryMP, Area, Timestamp,
SensorID, Latitude, Longitude
qos.sensorTypes =
double, double, double, double, double, double, double, double, string, bigint, strin
g, double, double
qos.lsmProperty =
http://lsm.der.i.e/OpenIoT/Temperature, http://lsm.der.i.e/OpenIoT/Humidity,
http://lsm.der.i.e/OpenIoT/Pressure, http://lsm.der.i.e/OpenIoT/NO2, http://l
sm.der.i.e/OpenIoT/SO2, http://lsm.der.i.e/OpenIoT/CO, http://lsm.der.i.e/Ope
nIoT/BatterySensor, http://lsm.der.i.e/OpenIoT/BatteryMobilePhone, http://lsm
.der.i.e/OpenIoT/Area, http://lsm.der.i.e/OpenIoT/Timestamp, http://lsm.der.i
.e/OpenIoT/SensorID, http://lsm.der.i.e/OpenIoT/Latitude, http://lsm.der.i.e/
OpenIoT/Longitude
qos.lsmUnit =
C, Percent, hPa, ug/m3, ug/m3, mg/m3, Percent, Percent, Unit, Unit, Unit, Unit
qos.testing = true
qos.logWriting = true

```

5.2 Developer Instructions

In the following section we are going to describe the steps required to build and deploy an OpenIoT module to JBoss application server.

5.2.1 System requirements

All you need to build the projects is Java 7.0 (Java SDK 1.7) or later and Maven 3.0 or above.

5.2.2 Download

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

- HTTPS: <https://github.com/OpenlotOrg/openiot.git>
- SSH: <git@github.com:OpenlotOrg/openiot.git>

5.2.3 Deploy from the source code

If you have not yet done so, you must Configure Maven before using any maven commands over OpenIoT modules. After that:

- To Start the 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. The following shows the 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 a module to JBoss:

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

 4. Make sure you have started the JBoss Server as described above.
 5. Open a command line and navigate to the root directory of the module's Project.
 6. Type this command to build and deploy the archive:
 - `mvn clean package jboss-as:deploy`
 7. This will deploy target/MODULE_NAME.war to the running instance of the server.
- Access the application:
 - The application will be running at the following URL:
`http://localhost:8080/MODULE_NAME/`.

- Un-deploy the Archive:
 2. Make sure you have started the JBoss Server as described above.
 3. Open a command line and navigate to the root directory of the module's Project.
 4. When you are finished testing, type this command to undeploy the archive:
 - `mvn jboss-as:undeploy`.

5.2.4 Run in Eclipse

5.2.4.1 Integrating and Starting JBoss server

You can start JBoss Application Server and deploy the 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 URL: <https://docs.jboss.org/author/display/AS7/Starting+JBoss+AS+from+Eclipse+with+JBoss+Tools>

5.2.4.2 Integrating and deploying of an OpenIoT MODULE

To integrate and deploy a 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 MODULE's source code directory that has been previously downloaded.
3. Choose the MODULE_NAME and click the Finish button.
4. Right click on the "MODULE_NAME" project and choose "Run As>Maven Build..."
5. Insert into:
 - a. Goals: "clean package jboss-as:deploy"
 - b. Profiles: "arq-jbossas-remote"
 - c. Name: "MODULE_NAME package-deploy" (or your preferred name)
6. Click the Run button (the JBoss Server should be already running). The project will be built and deployed automatically, and will run on the JBoss AS running instance. From now on this configuration should be available at the Eclipse Run Configurations under Maven Build.

To Undeploy the MODULE from the running instance of the JBoss AS, follow the steps below:

1. Right click on the "MODULE_NAME" project and choose "Run As>Maven Build..."
2. Insert into:
 - a. Goals: "jboss-as:undeploy"
 - b. Profiles: "arq-jbossas-remote"
 - c. Name: "MODULE_NAME 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.

5.3 User

5.3.1 System requirements

All you need to run an OpenIoT module project is Java 7.0 (Java SDK 1.7) or later, and JBoss Enterprise Application Platform 6 or JBoss AS 7.1.

You can download the binaries through the OpenIoT Wiki³³ under the Users>Downloads³⁴ section.

5.3.2 Deployment/Undeployment

5.3.2.1 JBoss AS 6.0

Deploy: To deploy the MODULE on the JBoss AS6.0, copy the “MODULE_NAME.war” to the server's “deploy” directory.

Undeploy: Remove the app war (MODULE_NAME.war) from the JBoss deploy directory while the server is running.

5.3.2.2 JBoss AS 7.0

Deploy: To deploy the MODULE on JBoss AS 7.0, copy the “MODULE_NAME.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 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>

³³ <https://github.com/OpenIoTOrg/openiot/wiki>

³⁴ <https://github.com/OpenIoTOrg/openiot/wiki/Downloads>

6 PLATFORM CONTAINERS, TOOLS & LIBRARIES USED

Since OpenIoT will be offered as open source project it is important that its various components use common libraries, platforms and development tools where applicable. This way it will be easier for the advanced users and developers to get involved and to move from one OpenIoT module to another without having to develop new skills to get involved. The main open source libraries, platform and development tools are listed below:

- Software project management tool:
 - **Apache Maven**³⁵. It can manage a project's build, reporting and documentation from a central piece of information (pom XML file). It will be used in the OpenIoT code as a library and build management tool.
- Development environment:
 - **Eclipse IDE**³⁶. It is one of the most popular open source Integrated Development Environments and will be used as the main development environment in the OpenIoT project.
- Web-Service implementation:
 - **RESTEasy**³⁷: is a JBoss project that provides various frameworks to help you build RESTful Web Services and RESTful Java applications. It is a fully certified and portable implementation of the JAX-RS specification. JAX-RS is a new JCP specification that provides a Java API for RESTful Web Services over the HTTP protocol.
- User interfaces: Some options for implementing the OpenIoT user interfaces are listed below:
 - Web Clients:
 - **JavaServer Faces (JSF)**³⁸: JavaServer Faces technology establishes the standard for building server-side user interfaces. With the contributions of the expert group, the JavaServer Faces APIs are being designed so that they can be leveraged by tools that will make web application development even easier.

³⁵ <http://maven.apache.org/>

³⁶ <http://www.eclipse.org/>

³⁷ <http://www.jboss.org/resteasy>

³⁸ <http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>

- **PrimeFaces**³⁹: Prime Technology is not a software vendor but a software development house along with the consulting and training activities. A framework that is not even used by its own creators can easily miss vital points regarding usability and simplicity. A major difference compared to vendor products is that we use PrimeFaces in all of our clients' projects as the front end framework. This helps us to view the project from an application developer's point of view so that we can easily realize the missing features and quickly fix the bugs. This significantly distinguishes PrimeFaces from other libraries.
- **jsPlumb**⁴⁰: provides means for a developer to visually connect elements on their web pages. It uses SVG or Canvas in modern browsers and VML on IE 8 and below.
- Platform Management & Monitoring:
 - **Java Management Extensions (JMX)**. The JMX technology provides tools for building distributed, Web-based, modular and dynamic solutions for managing and monitoring devices, applications and service-driven networks. The JMX technology could be used to monitor and manage different OpenIoT modules from the OpenIoT Configuration/Monitor console.
 - **JavaMelody**⁴¹: The goal of JavaMelody is to monitor Java or Java EE application servers in quality assurance and production environments.
- Enterprise Application Platform. OpenIoT, at this point, will use the JBoss Application platform to serve as an Enterprise Server container to host the various OpenIoT modules.
 - **JBoss Application Platform**⁴². The JBoss Application Platform was created with the cloud in mind. It is based on a services-driven set of components and is running OSGi and the Java EE application server side by side.
- RDF Database. OpenIoT will use LSM as the RDF store which uses the commercial version of Virtuoso and will also support the following Open Source database:

³⁹ <http://www.primefaces.org/>

⁴⁰ <https://github.com/sporritt/jsplumb/>

⁴¹ <http://code.google.com/p/javamelody/>

⁴² <http://www.redhat.com/products/jbossenterprisemiddleware/application-platform/>

- **Sesame**⁴³. Framework for querying and analysing RDF data. This includes parsing, storing, inferring and querying of/over such data. It offers an easy-to-use API that can be connected to all leading RDF storage solutions.
- Testing Framework.
 - **JUnit**⁴⁴: is a unit testing framework for the Java programming language.
- SPARQL Client.
 - **Sesame**⁴⁵: For accessing SPARQL interfaces Sesame client is used.
- Logging.
 - **Logback**⁴⁶: Logback is intended as a successor to the popular log4j project, picking up where log4j leaves off. Logback's architecture is sufficiently generic so as to apply under different circumstances. At present time, logback is divided into three modules, logback-core, logback-classic and logback-access.

6.1 OpenIoT Components-Libraries relation

In this section we list all the libraries and their version that different OpenIoT components use.

6.1.1 Scheduler

The Scheduler libraries and containers are listed in Table 13 below.

Table 13: Scheduler libraries and containers

Container/Library Name	Version
JBoss	7.1.x
RESTEasy	2.3.6
Apache Maven	3.x.x
Logback	1.0.11
JUnit	
Jboss maven plugin	7.3
Sesame	2.7.0

⁴³ <http://www.openrdf.org/>

⁴⁴ <https://github.com/junit-team/junit>

⁴⁵ <http://www.openrdf.org/>

⁴⁶ <http://logback.qos.ch/>

Java	1.6+
(OpenIoT) utils.commons	0.0.1
org.eclipse.persistence.eclipselink	2.5.0
JavaMelody	1.45.0
com.hp.hpl.jena	2.6.4
lsm.api	0.0.1
org.openrdf.sesame	2.7.0

6.1.2 Service Delivery & Utility Manager

The SD&UM's libraries and containers are listed in Table 14 below.

Table 14: SD&UM's libraries and containers

Container/Library Name	Version
JBoss	7.1.x
RESTEasy	2.3.6
Apache Maven	3.x.x
Logback	1.0.11
JUnit	
Jboss maven plugin	7.3
Sesame	2.7.0
Java	1.6+
(OpenIoT) utils.commons	
JavaMelody	1.45.0
org.eclipse.persistence.eclipselink	2.5.0
com.hp.hpl.jena	2.6.4
lsm.api	0.0.1
org.openrdf.sesame	2.7.0

6.1.3 X-GSN

The X-GSN's libraries and containers are listed in Table 15 below.

Table 15: X-GSN's libraries and containers

Container/Library Name	Version
ant	1.7.0
antlr	2.7.7
asterisk-java	0.3
axis2	1.4.1
com.typesafe.config	1.2.0
commons-codec	1.3
commons-dbcp	1.4
commons-email	1.2

commons-fileupload	1.2
commons-io	2.1
commons-lang	2.2
commons-math	1.2
cos	05Nov2002
easymock	2.5.1
easymockclassexension	2.2
groovy-all	1.7.1
h2	1.1.116
httpclient	4.0.1
httpcore	4.0.1
httpcore-nio	4.0.1
httpmime	4.0.1
httpunit	1.6.2
jansi	1.10
jasperreports	3.0.0
Java	1.7
jetty-all-server	7.0.2.v20100331
jfreechart	1.0.14
jibx-run	1.2.5
lbx-extras	1.2.5
joda-time	1.6
json-simple	1.1
jts	1.8
junit	4.10
layout	x
log4j	1.2.15
logback-core	1.1.2
lsm	j6_2013-06-28
mina-core	2.0.7
mysql-connector-java	5.1.26
net.tinyos	1.x
opencsv	1.8
org.apache.commons.collections	3.2.1
postgis-jdbc	1.3.0
REngine	0.6-1
Resteasy-jaxrs	3.0.8
rome	0.9
Rserve	0.6-1
rxtx	2.1.7
stringtemplate	3.0
tinyos-java	2.1
Twitter	2.x
webcam-capture-driver-jmf	0.3.9
xercesImpl	2.8.1
xstream	1.3.1

6.1.4 LSM-Light

The LSM-Light's libraries and containers are listed in Table 16 below.

Table 16: LSM's Light libraries and containers

Container/Library Name	Version
Java	1.7
Apache Commons	1.8.2
Virtuoso Jena Provider	2.6.2
com.hp.hpl.jena	2.6.4
JBoss	7.1.x
Log4j	1.2.14
Dom4j	1.6.1
Jaxen	1.1 beta
Saxon9he	
jUnit	3.8.1
JFreeChart	1.0.6
HttpClient	4.0
Bonecp	0.7.1
Virtjdbc	4
Json	20090211
OpenIoT utils.commons	0.0.1

6.1.5 Security Module

The Security Module libraries and containers are listed in **Table 17** below.

Table 17: Security Module libraries and containers

Container/Library Name	Version
Java	1.7
JBoss	7.1.x
Jasig CAS	3.5.2
inspektr	1.0.7.GA
person-directory	1.5.1
commons-collections	3.2.1
commons-codec	1.4
Spring Framework	3.1.1.RELEASE
not-yet-commons-ssl	0.3.9
org.restlet	1.1.1
OpenIoT utils.commons	0.0.1
sojo	1.0.5

xstream	1.3
xpp3_min	1.1.4c
xmltooling	1.3.2-1
xmlsec	1.4.3
validation-api	1.0.0.GA
velocity	1.5
wstx-asl	3.2.9
xalan	2.7.1
xercesImpl	2.10.0
Apache Shiro	1.2.2
shiro-faces	2.0-SNAPSHOT
slf4j-api	1.7.1
slf4j-log4j12	1.7.1
serializer	2.7.1
pac4j	1.4.1
buji-pac4j	1.2.1
perf4j	0.9.16
quartz	1.6.1
scannotation	1.0.3
scribe	1.3.2
scribe-up	1.2.0
OpenIoT security.client	0.0.1-SNAPSHOT
servlet-api	2.5
httpclient	4.1.2
httpcore	4.1.2
icu4j	3.4.4

6.1.6 CUPUS middleware

The Security Module libraries and containers are listed in Table 18 below .

Table 18: CUPUS middleware libraries and containers

Container/Library Name	Version
Java	1.7
JUnit	
onejar-maven-plugin	1.4.4

6.1.7 QoS manager

The Security Module libraries and containers are listed in Table 19 below.

Table 19: QoS Manager libraries and containers

Container/Library Name	Version
Java	1.7
JUnit	
onejar-maven-plugin	1.4.4

6.1.8 User Interface

6.1.8.1 Request Presentation

The Request Presentation's libraries and containers are listed in Table 20 below.

Table 20: Request Presentation's libraries and containers

Container/Library Name	Version
Java	1.6+
Json	20090211
Primefaces	3.5
Primefaces-extensions	0.6.3
Primefaces-extensions (codemirror addon)	0.6.3
Apache commons-lang3	3.0
Jsf-api	2.1.20
El-api	2.2
Jaxb-api	2.1
Jaxb-impl	2.1
Resteasy-jaxrs	2.3.1.GA
Javamelody-core	1.45.0
OpenIoT utils.commons	0.0.1
OpenIoT utils.requestCommons	0.0.1

6.1.8.2 Request Definition

The Request Definition's libraries and containers are listed in Table 21 below.

Table 21: Request Definition's libraries and containers

Container/Library Name	Version
Java	1.6+
Json	20090211
Primefaces	3.5
Primefaces-extensions	0.6.3
Apache commons-lang3	3.0
Apache commons-io	1.3.2

Jsf-api	2.1.20
El-api	2.2
Jaxb-api	2.1
Jaxb-impl	2.1
Resteasy-jaxrs	2.3.1.GA
Javamelody-core	1.45.0
JsPlumb	1.4.0
OpenIoT utils.commons	0.0.1
OpenIoT utils.requestCommons	0.0.1

6.1.8.3 Schema Editor

The Schema Editor libraries and containers are listed in Table 22 below.

Table 22: Schema Editor libraries and containers

Container/Library Name	Version
Java	1.7
junit	4.11
org.primefaces	3.5
org.primefaces.extensions	0.701
jstl	1.2
JBoss	7.1.x
atmosphere-runtime	1.0.0.RC1
jsf-api	2.1.7
jsf-impl	2.1.7
javax.faces-api	2.2
servlet-api	2.5
jsp-api	2.1
slf4j-log4j12	1.7.5
org.restlet.jse	2.0.14
org.restlet.ext.json	2.0.14

7 OPENIOT END TO END DEMONSTRATION

A fully deployable service (from data Capturing to Visualization) using OpenIoT reference framework is explained in this section. This section serves also as a tutorial of setting up and presenting a service with the help of OpenIoT platform.

7.1 Data Capturing and Flow description

In this example, weather sensors are deployed in the central area of Brussels producing data (wind chill temperature, atmospheric pressure, air temperature, atmosphere humidity and wind speed).

The data are capture from X-GSN through a special wrapper (the physical plane of Figure 6 below) which collects the Weather Station's data every 4 hours. This is where the first level of data filtering occurs, whereas the weather station produces data in a higher rate, in our scenario we are interested in a four hour sampling rate. The captured data are following a sensor type created for this occasion that we have named "Weather". The "Weather" sensor type is used to semantically annotate the captured data at the X-GSN level. One X-GSN instance is running for every weather station so after X-GSN announces the existence of each sensor (bound with a specific sensor id) it starts to push the captured data to LSM-Light (RDF Store), which is deployed at DERI⁴⁷ premises in Galway (the virtualized plane of Figure 6 below)⁴⁸.

Then it is time to set up the service by using the Request Definition (the utility/application plane of Figure 6 below) tool with the help of which we will discover these sensors (by using the Scheduler), describe the request and send it to the Scheduler (the virtualized plane of Figure 6 below) to handle it. The Scheduler decomposes the request and registers it to LSM-light. The information that we want to get for this scenario is the wind chill temperature versus the actual air temperature in the area of Brussels for the dates between 01/07/2014 and 01/28/2014.

The SD&UM (the virtualized plane of Figure 6 below) retrieves on demand the formulated request executes the involved queries and feeds the Request Presentation (the utility/application plane of Figure 6 below) with presentation data. The last step would be for the Request Presentation to presents the received data in the predefined widgets.

Now that we had a high level description of the data flow at the virtualized and utility/application plane of Figure 6 below let's build it and present it step by step below.

⁴⁷ <http://www.deri.ie/>

⁴⁸ <http://lsm.deri.ie/>

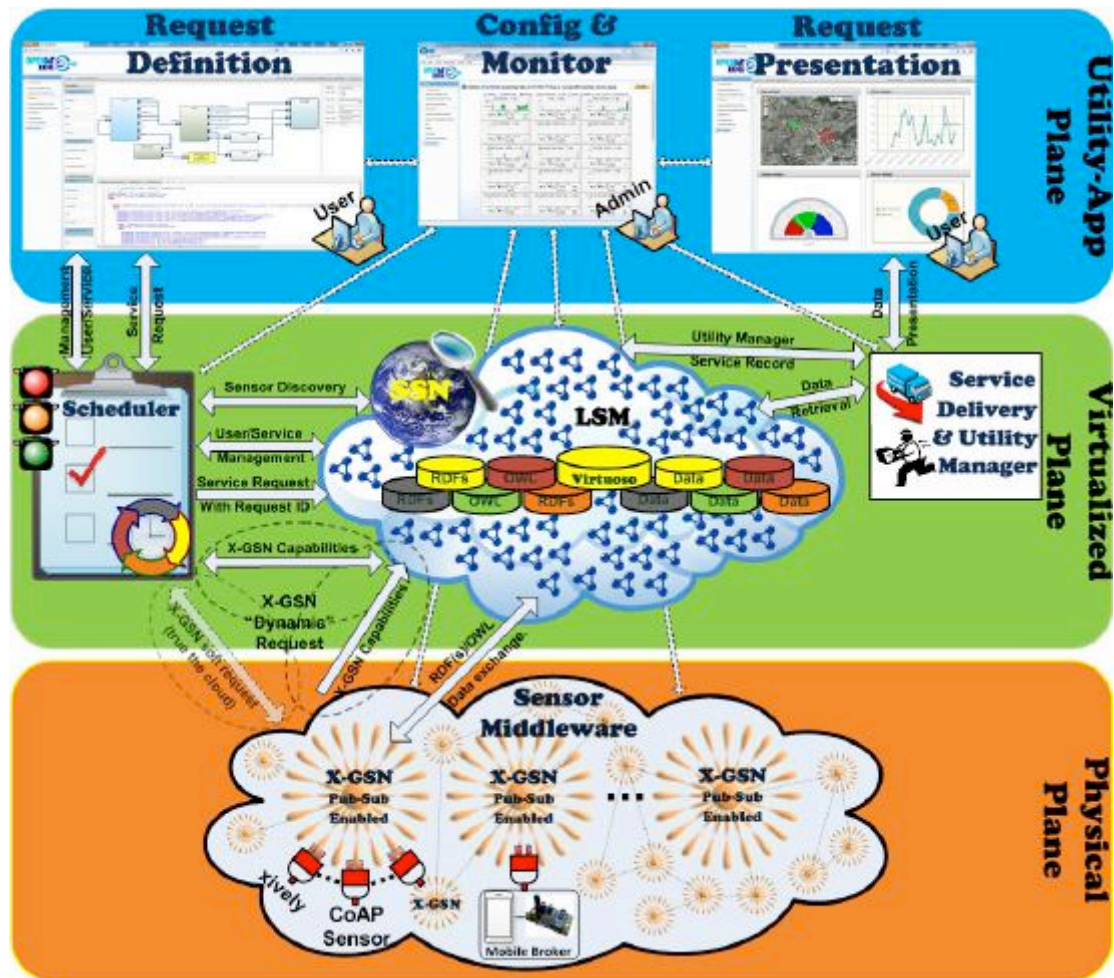


Figure 6: OpenIoT Architecture.

7.2 Setting up the experiment

7.2.1 Edge Server setup (X-GSN)

Once the sensor description has been created, we need to configure a virtual sensor in X-GSN that corresponds to that description, and which gathers the data from the “Weather” sensor deployed in Brussels.

First of all, we need to make sure that X-GSN is properly configured and connected to LSM. In order to connect GSN with LSM, you need to edit the file *application.conf*. There, you need to specify the X-GSN CAS credentials and LSM schema to use. See the following sample:

```
username=gsnuser
password=gsnpass
metaGraph="http://lsm.der.i.e/OpenIoT/sensormeta#"
dataGraph="http://lsm.der.i.e/OpenIoT/sensordata#"
lsm.server=" http://lsm.der.i.e/lsm-light.server/"
```

7.2.1.1 Semantic annotation of sensor data

In order to associate metadata with a virtual sensor, you need to create a metadata file that will be associated with the virtual sensor. The metadata file has to be located in the same folder as the virtual sensor and needs to have the same name as the virtual sensor name but with the extension (*.metadata*). For example, a virtual sensor named `Brussels_weather.xml` will have an associated metadata file named `Brussels_weather.metadata`. The metadata file contains information such as the location (in coordinates), as well as the fields exposed by the virtual sensor. This also includes the mapping between a sensor field (e.g. `airtemperature`) and the corresponding high-level concept of the ontology (e.g. `http://openiot.eu/ontology/ns/AirTemperature`).

Table 23: Weather metadata file

```
sensorID="http://lsm.der.i.e/resource/61330620147099"
sensorName=979128
source="Brussels netatmo"
sensorType=weather
information=Weather sensors in Brussels
author=openiot
feature="http://lsm.der.i.e/OpenIoT/BrusselsFeature"
fields="pressure,airtemperature,humidity,visibility,windchill,windspeed"
field.airtemperature.propertyName="http://openiot.eu/ontology/ns/AirTemperature"
"
field.airtemperature.unit=C
field.humidity.propertyName=" http://openiot.eu/ontology/ns/AtmosphereHumidity"
field.humidity.unit=Percent
field.visibility.propertyName="
http://openiot.eu/ontology/ns/AtmosphereVisibility"
field.visibility.unit=Percent
field.pressure.propertyName="
http://openiot.eu/ontology/ns/AtmosphericPressure"
field.pressure.unit=mb
field.windchill.propertyName=" http://openiot.eu/ontology/ns/WindChill"
field.windchill.unit=C
field.windspeed.propertyName=" http://openiot.eu/ontology/ns/WindSpeed"
field.windspeed.unit=Km/h
latitude=51.33332825
longitude=3.200000048
```

7.2.1.2 Registering sensors to LSM

Sensors can be registered to LSM by executing the script *lsm-register.sh* (on Linux/Mac) or *lsm-register.bat* (on Windows). This script takes as argument the metadata file name. After this, the corresponding metadata in RDF will have been stored in LSM.

Example:

```
./lsm-register.sh virtual-sensors/brussels_weather.metadata
lsm-register.bat virtual-sensors\brussels_weather.metadata
```

7.2.1.3 Pushing data to LSM

In order to push data to LSM, the LSMExporter processing class is internally used by X-GSN. This is specified in the virtual sensor configuration file:

```
<processing-class>
  <class-name>org.openiot.gsn.vsensor.LSMExporter</class-name>
  <init-params>
    <param name="allow-nulls">false</param>
    <param name="publish-to-lsm">true</param>
  </init-params>
  <output-structure>
    <field name="airtemperature" type="double" />
    <field name="humidity" type="double" />
    <field name="pressure" type="double" />
    <field name="windspeed" type="double" />
    <field name="windchill" type="double" />
    <field name="visibility" type="double" />
  </output-structure>
</processing-class>
```

Then, when X-GSN starts, it begins to acquire the data through the wrapper and automatically generating the RDF data for each observation, storing it in LSM.

Each observation will be assigned a unique URI, e.g.

<http://lsm.derri.ie/resource/29925179667811>

Then, you can query the Virtuoso server to see the updated data: e.g.:

```
select * where {
  <http://lsm.derri.ie/resource/29925179667811> ?p ?o
}
```

And get the results:

http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://purl.oclc.org/NET/ssnx/ssn#Observation
http://purl.oclc.org/NET/ssnx/ssn#observedBy	http://lsm.derri.ie/resource/29855158254802
http://purl.oclc.org/NET/ssnx/ssn#observationResultTime	2013-05-15T11:45:00Z
http://purl.oclc.org/NET/ssnx/ssn#featureOfInterest	http://lsm.derri.ie/resource/3797289123726234

Once the data is in LSM, it can be accessed by the other OpenIoT components.

7.3 Service Setup

From this point the same steps as described in deliverable D3.3.2 (chapter 6.2) will be followed for building and visualizing the request.

7.3.1 Building the Request

The first step, for building the request, would be to log in to the Request Definition by using our credentials (Figure 7 below).



Figure 7: Request Definition log-in.

By logging in our profile is loaded and all our previously defined services are available to view or edit (Figure 8 below).

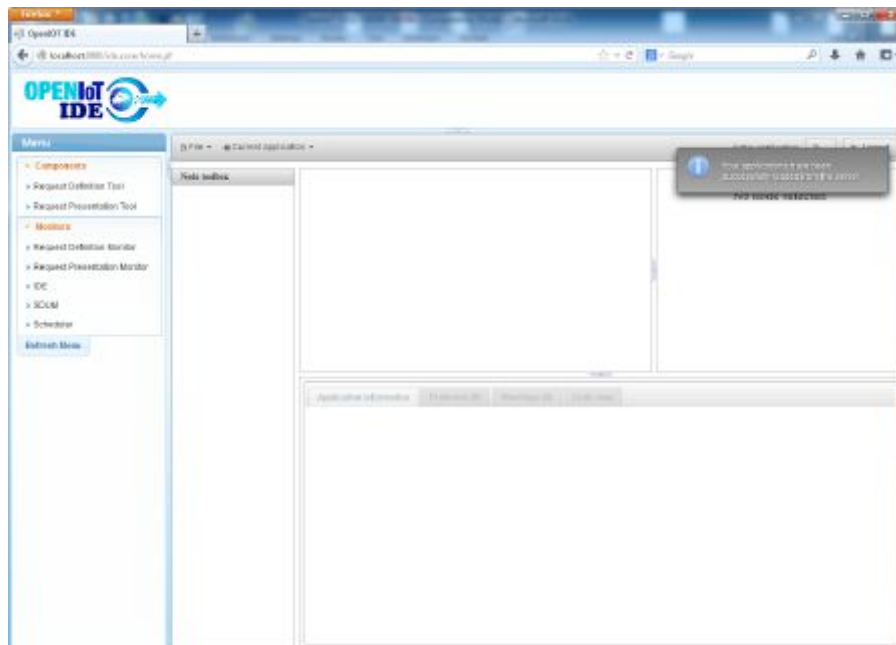


Figure 8: Request Definition loaded profile.

By going to the “File” menu we can create a new Application (Figure 9 below).

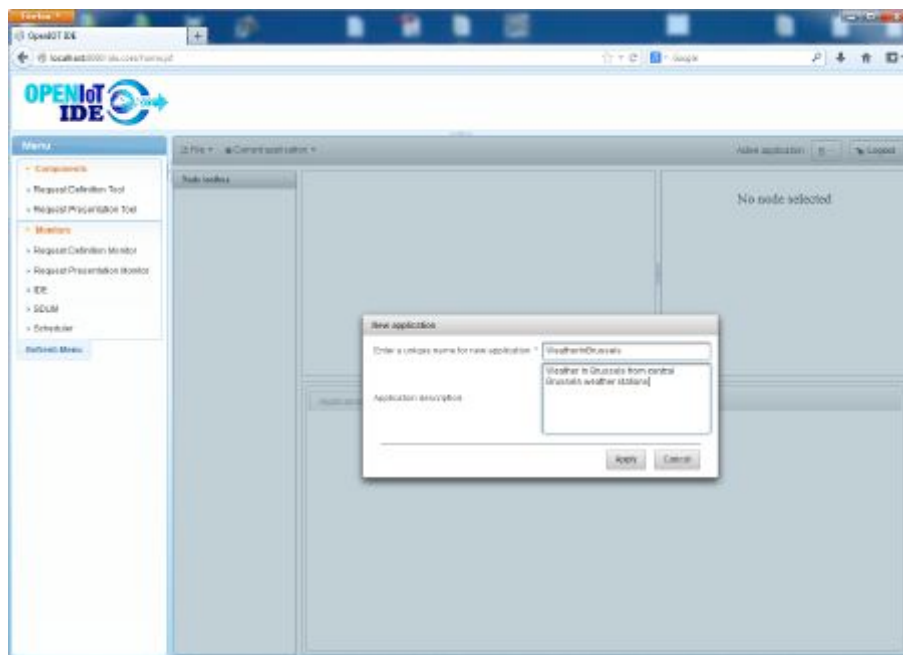


Figure 9: New application creation.

The first thing we need to do is to discover the available sensors. This is done by hitting the magnifying glass at the Data sources toolbox (Figure 10 below).

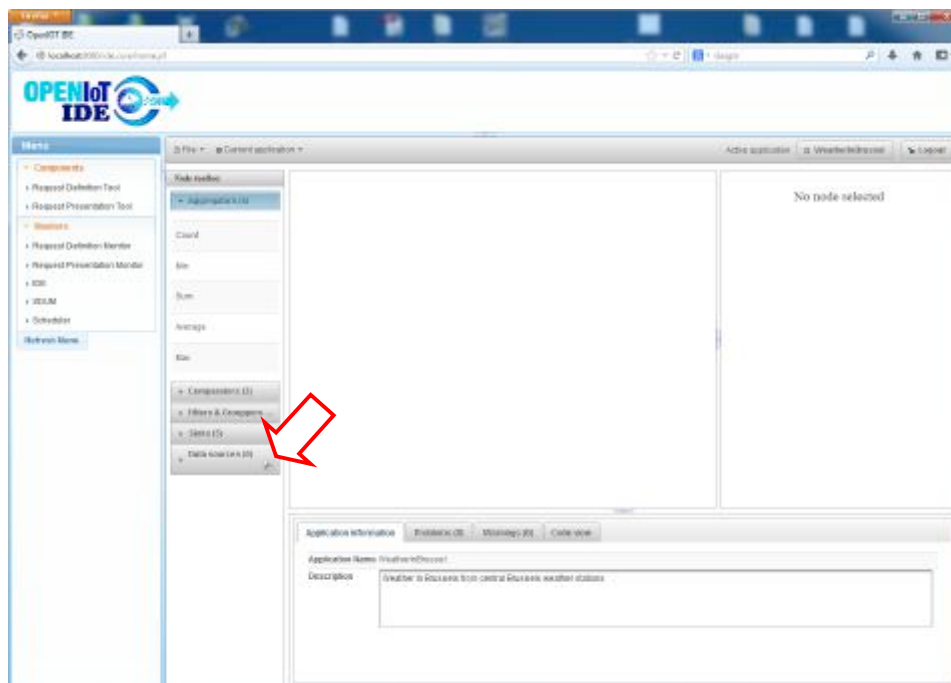


Figure 10: Discover sensors button.

In the map that appears we look up for the Brussels area and we add a pinpoint to the map. Then we set the radius of interest and we hit the “Find sensors” button (Figure 11 below)

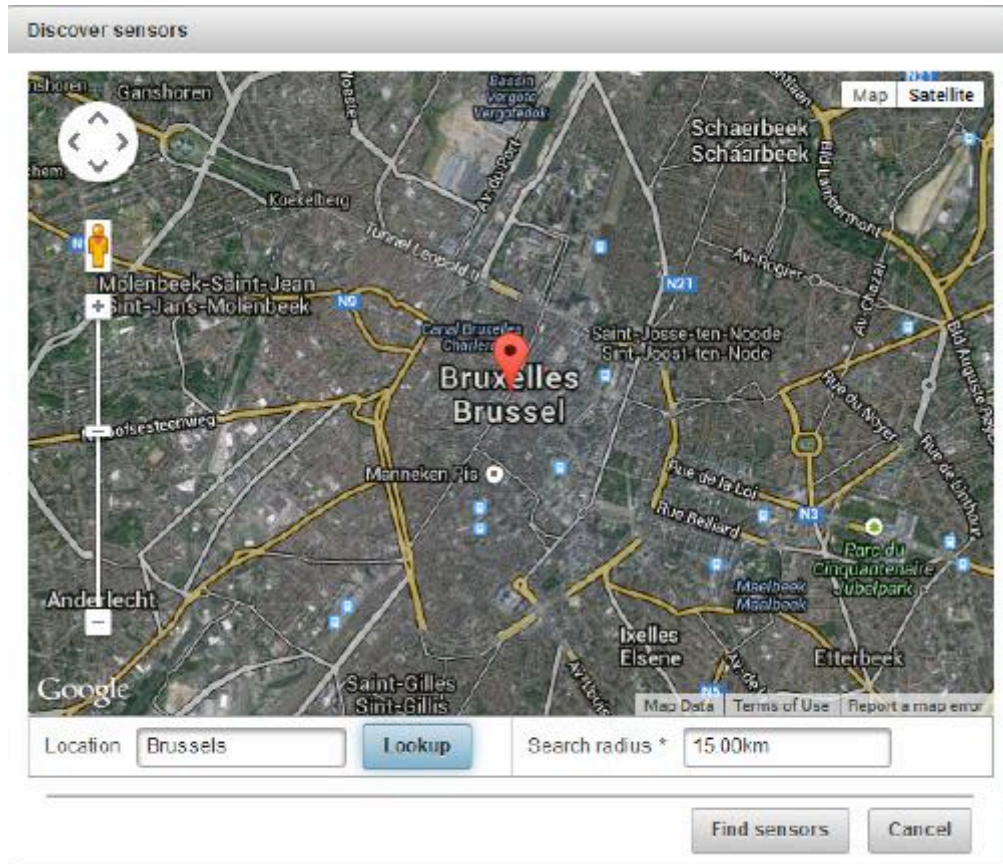


Figure 11: Discover Sensors in Brussels

This request is sent to the Scheduler that in its turn queries LSM-Light for available sensors in this area. The reported, from LSM-Light, sensor types are sent to the Scheduler that in its turn sends to the Request Definition so as to fill the available “Data sources” toolbox (Figure 12 below). As we can see two sensor types are deployed in that area (weather and ITK). By dragging and dropping the blocks from our toolbox we start to build our request (Figure 12 below). We drag and drop the “weather” sensor type and as we can see all the sensor type observations (outputs) are available to interact with (wind chill temperature, atmospheric pressure, air temperature, atmosphere humidity and wind speed).

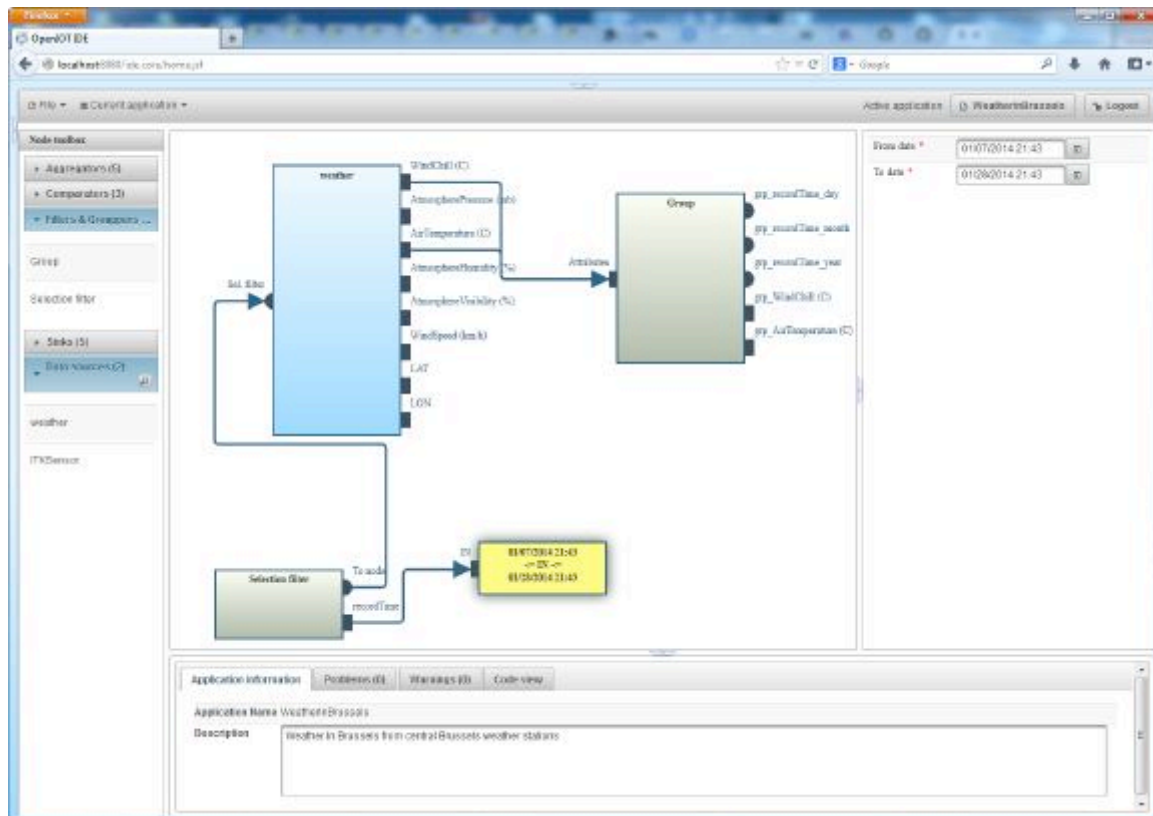


Figure 12: Comparator (Between) properties.

We require a “Selection filter” from our “Filters & Groupers” toolbox that we connect one side of it with the node and the other one with a “Between” comparator that we have already dropped to our workspace from the “Comparators” toolbox. We set up the “Between” comparator between “01/07/2014” and “01/28/2014” (three weeks) which are the dates of interest to us to collect our data (Figure 12 above).

Our next step is to add a “Group” node from the “Filters & Groupers” toolbox which we are going to use so as to group the Wind Chill and Air Temperature by Year/Month/Day which is selected through the node’s options (Figure 13 below). We connect the Wind chill and Air temperature outputs of the “weather” node to the “Group” node attributes and automatically, as we can see in Figure 12 above, these outputs are generated also to the “Group” node.

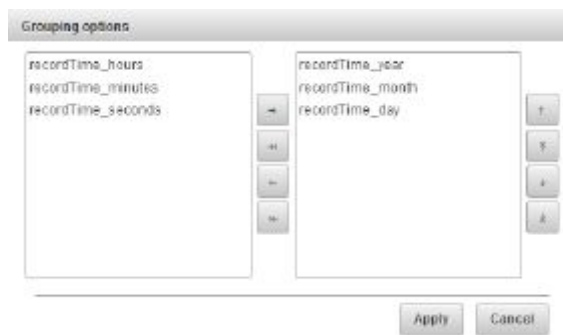


Figure 13: Grouping options.

Since we need the average values for every day, we drag and drop two “Average” nodes from the “Aggregators” toolbox to our workspace and we connect the Wind Chill and Air Temperature outputs to them respectively (see Figure 14 below).

We are going to visualise our output (the two average values for every day) to a line chart so the next step is to drag and drop a “Line Chart” from the “Sinks” toolbox. We are going to use the X axis to present the time and the Y axis to present/compare the temperature values. At the line chart properties, see Figure 14 below, we select two series count (since we need to visualize two inputs) and for the X axis we select date observation as type. So we connect all the day/month/year output of the “Group” node to “x1” and “x2” inputs of the “Line Chart” node respectively and the Wind Chill and Air Temperature outputs to “y1” and “y2” inputs respectively (see Figure 14 below).

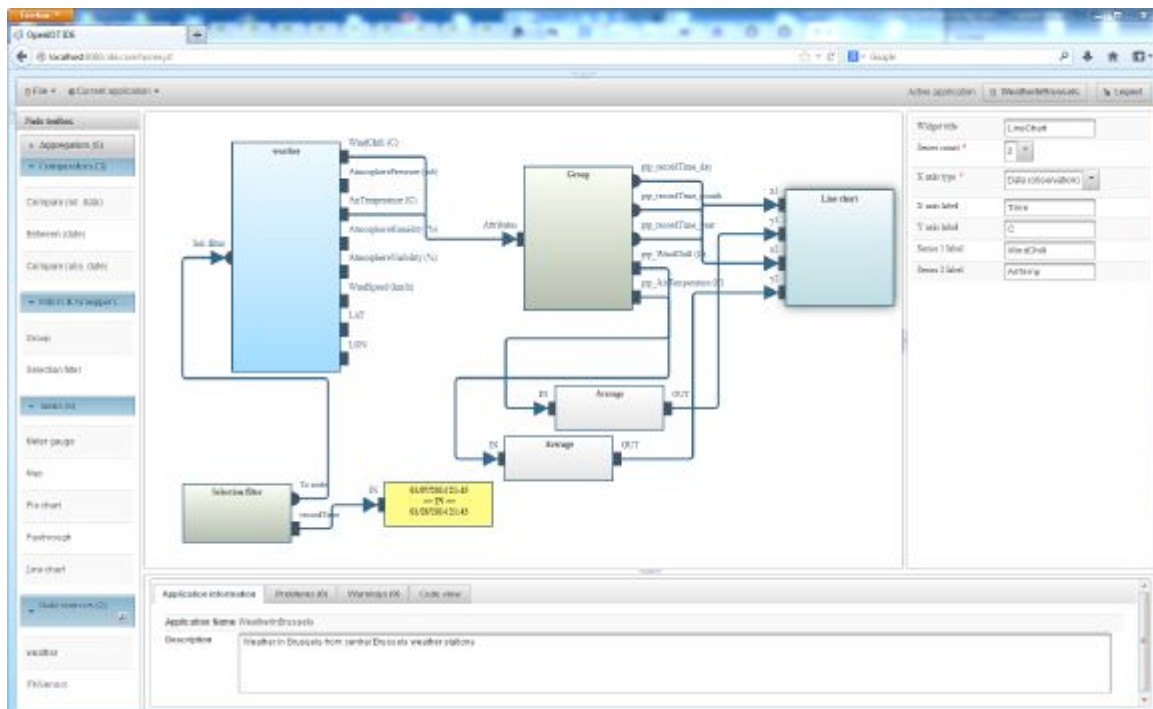


Figure 14: Line chart properties.

After having finished building the service we go to the “Current application” menu and hit the “Validate design” button (see Figure 15 below).

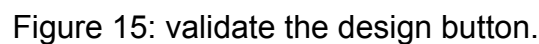
[illegible]

Figure 16: SPARQL script generation.

For testing purposes these scripts could be taken and executed directly against the SPARQL interface of LSM (see Figure 17 below) running, in our case, at DERI (<http://lsm.deri.ie/sparql>). For the Wind Chill temperature if we execute the script generated from the Request definition tool (available in Figure 16 above) and hit the “Run Query” button (see Figure 17 below) we get the mean wind chill temperature for the days between 15-27 of January 2014 (see Figure 18 below).

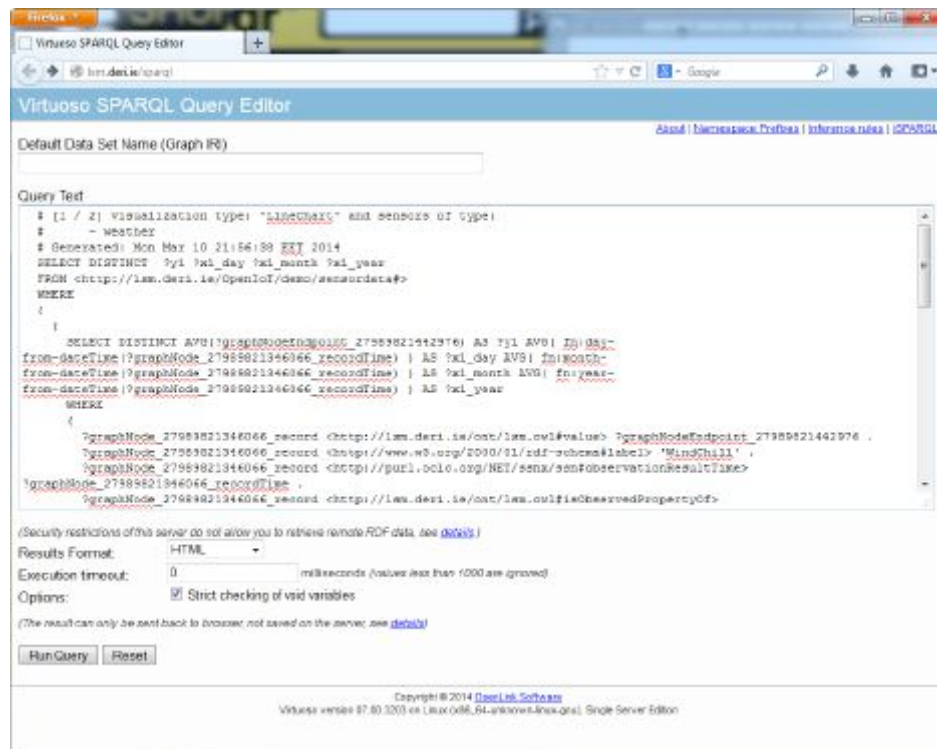


Figure 17: LSM-Light SPARQL endpoint (2 weeks Wind Chill in Brussels).

y1	x1_day	x1_month	x1_year
1.25	15	1	2014
4.666666666666667	16	1	2014
4.041666666666667	17	1	2014
4.5	18	1	2014
6	19	1	2014
2.958333333333333	20	1	2014
2.541666666666667	21	1	2014
-1.291666666666667	22	1	2014
0.535714285714286	23	1	2014
0.333333333333333	24	1	2014
-0.333333333333333	25	1	2014
1.666666666666667	26	1	2014
-1.333333333333333	27	1	2014

Figure 18: LSM-Light SPARQL endpoint result (2 weeks Wind Chill in Brussels).

For the Air temperature if we execute the script generated from the Request definition tool (available in Figure 16 above) and hit the “Run Query” button (see Figure 19 below) we get the mean air temperature for the days between 15-27 of January 2014 (see Figure 62 below).

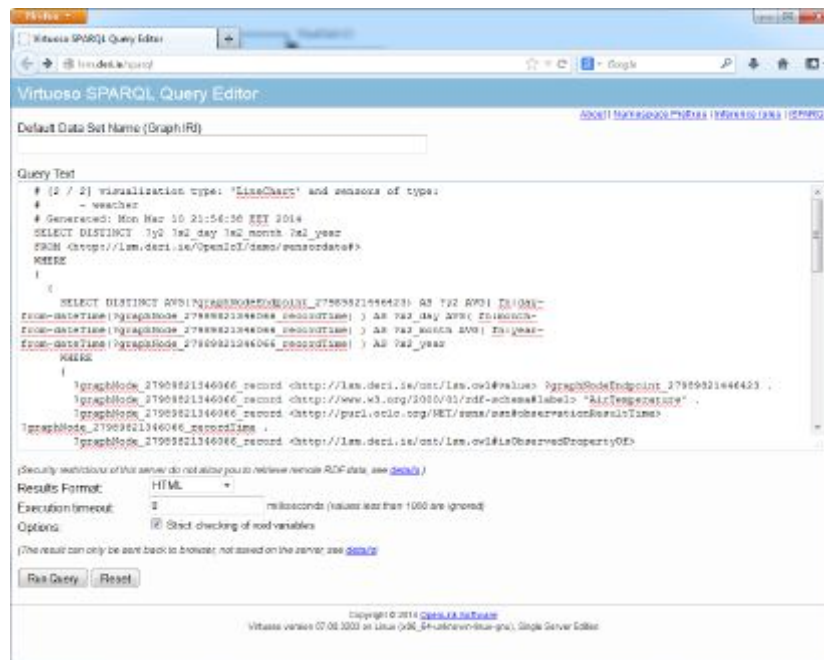


Figure 19: LSM-Light SPARQL endpoint (2week Air temp in Brussels).

y2	x2_day	x2_month	x2_year
5.5	15	1	2014
7.833333333333333	16	1	2014
7.708333333333333	17	1	2014
6.833333333333333	18	1	2014
7.666666666666667	19	1	2014
5.541666666666667	20	1	2014
4.041666666666667	21	1	2014
2.5	22	1	2014
4.178571428571429	23	1	2014
2.833333333333333	24	1	2014
3.833333333333333	25	1	2014
5.416666666666667	26	1	2014
3.666666666666667	27	1	2014

Figure 20: LSM-Light SPARQL endpoint result (2week Air temp in Brussels).

Finally, back to the Request Definition UI, we need to save (register) our newly described application to the Scheduler. So by going to the “Current application” menu and hitting “Save application” button (see Figure 21 below) we call the “registerService (osdSpec: OSDSpec): String” rest service of the scheduler where we provide the OSDSpec object generated from the process described above.

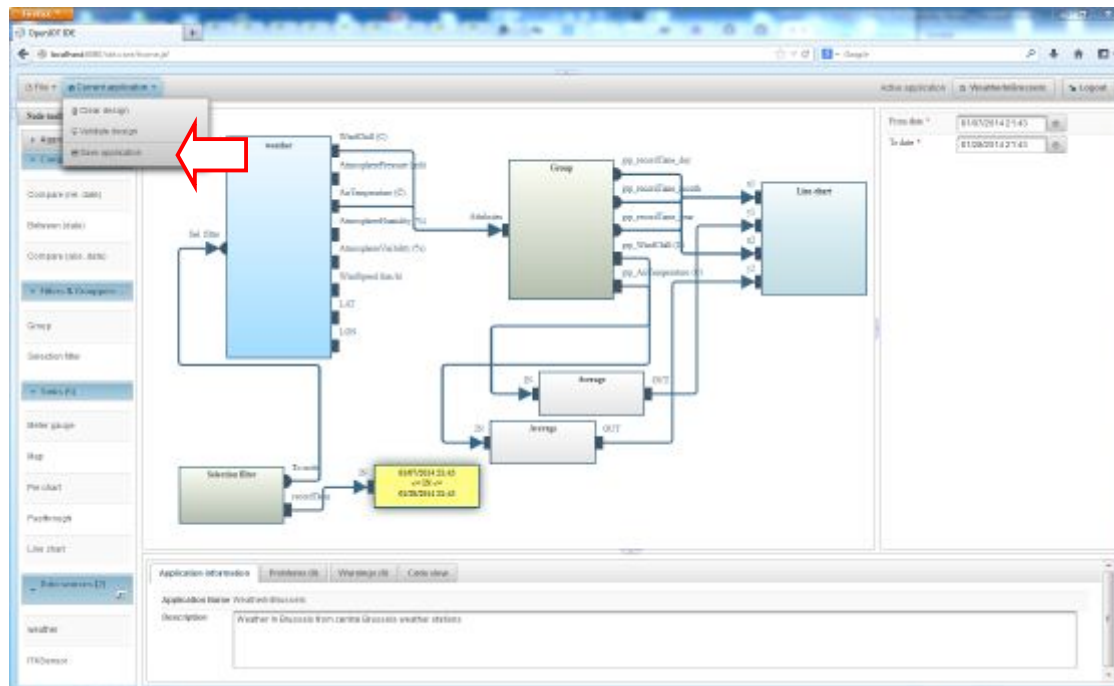


Figure 21: Save application button.

7.4 Visualising the Request

Now that we have finished the definition and registration of the service we can proceed to the visualisation of the captured data. So we log-in with our credentials to the Request Presentation UI (see Figure 22 below).

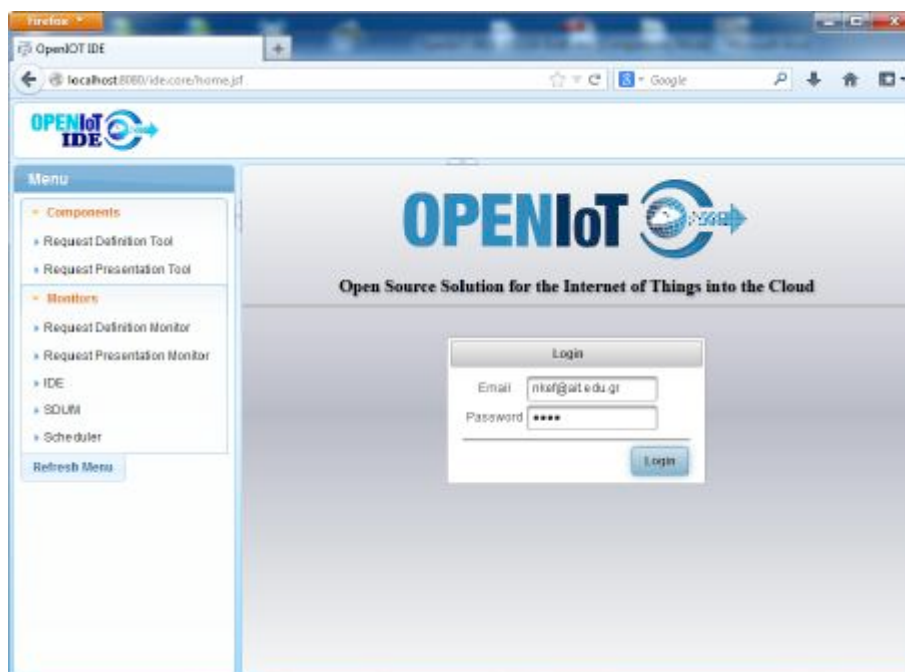


Figure 22: Request Presentation log-in.

And our profile is loaded. This means that the Request Presentation by using our credentials request from the SD&UM rest service to provide all the registered services from that user. The SD&UM in its turn builds the appropriate scripts to query this information from LSM-Light and provide the replied list to the Request Presentation module (see Figure 23 and Figure 24 below).

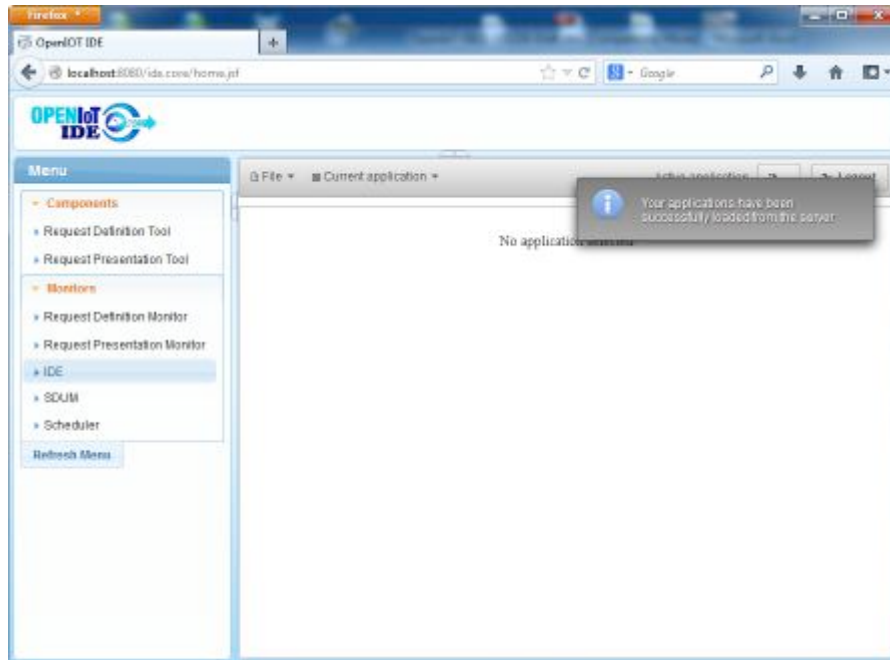


Figure 23: Request Presentation loaded profile.

Then we choose the application of interest to us, which in our case is the “WeatherInBrussels”.

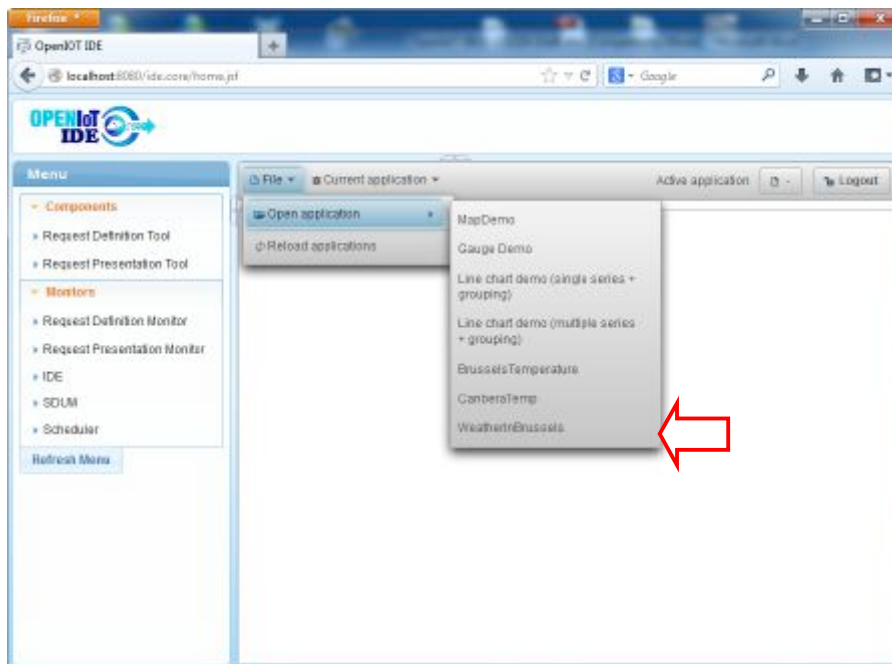


Figure 24: Load “WeatherInBrussels” scenario.

And we can see an empty widget related with that application to appear, which is the line chart diagram (see Figure 25 below).

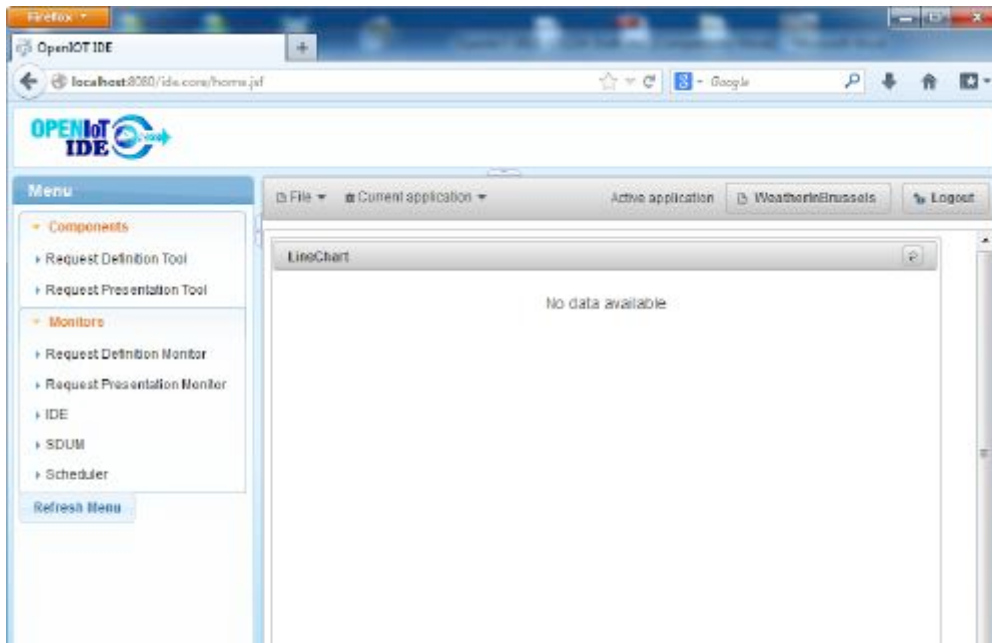


Figure 25: scenario loaded.

We need to hit the “force dashboard refresh” button from the “Current application” menu (Figure 26 below) so as the Request Presentation to use the “pollForReport (serviceID: String): SdumServiceResultSet” rest service of the SD&UM.

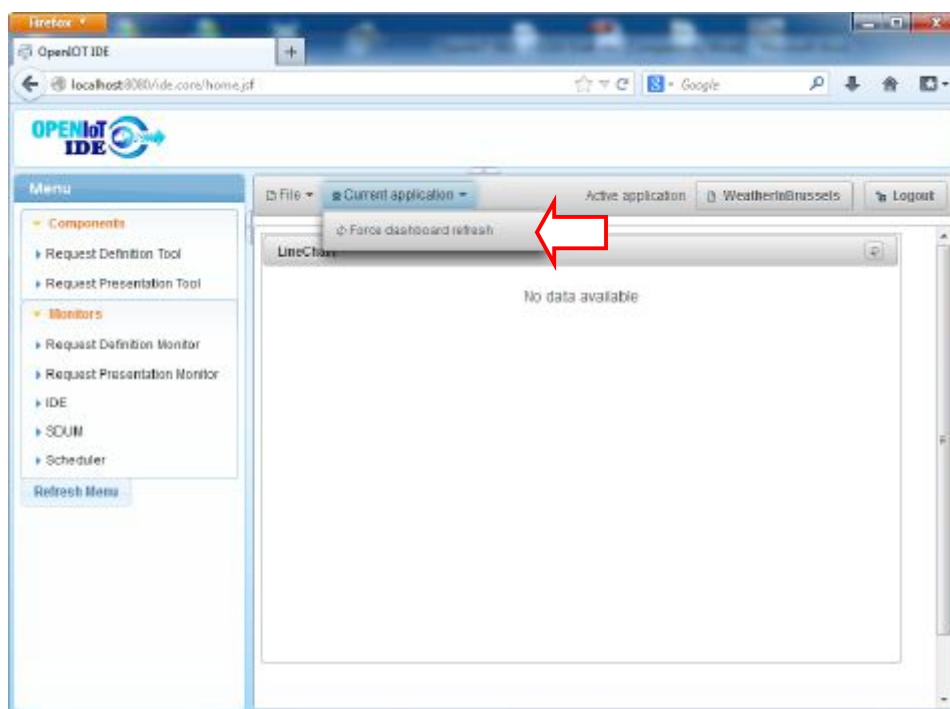


Figure 26: refresh the dashboard.

This SD&UM service retrieves the previously registered application from the LSM-Light module, retrieves the involved SPARQL scripts, executes them against the LSM-Light SPARQL interface, analyses the results, builds a SdumServiceResultSet which includes a list of the results and how to present them to the widget and finally sends these data to the Request Presentation module where we can see our result visualized (see Figure 27 below). So as we can see at the Line Chart widget, in Figure 27 below, we get a filtered result set from the initially raw data stored to our database every 4 hours of the average Wind Chill temperature versus average Air temperature between 15th and 27th of January 2014 in Brussels.

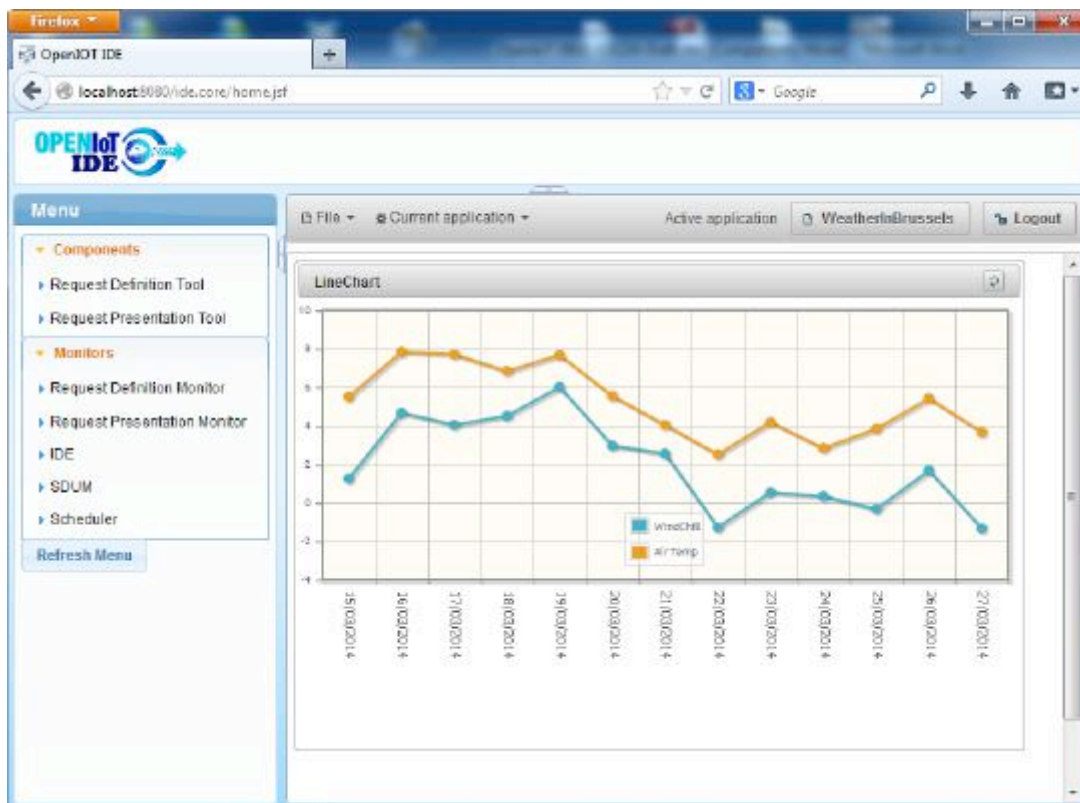


Figure 27: Wind Chill VS Air temperature in Brussels line chart.

8 CONCLUSIONS

Deliverable D4.3.2 corresponds to the second official release of the open source implementation of the OpenIoT middleware. The release comprises the prototype implementation of the major components of the OpenIoT architecture, along with accompanying documentation which is provided as part of this report. The open source implementation of the above mentioned components in the release is available within the Github infrastructure of the project (<https://github.com/OpenIoTOrg/openiot>).

This report includes a thorough description of the technical implementation of the key OpenIoT components, including the Scheduler, Service Delivery Manager, Link Sensor Middleware data platform, User Interfaces that relate to the definition and presentation of OpenIoT services, CUPUS, Security, as well as extended Global Sensor Networks middleware. The description includes internal component structures, their interfaces, and documentation for both end-users and developers/integrators. Furthermore, a description of the required third-party components and libraries is included.

The second OpenIoT middleware release includes comprehensive functionality for injecting sensor data into a cloud infrastructure, and for creating services and applications using data from the sensor cloud. This middleware release constitute one of the major integrated results of the project, which bundles together several technical developments of the project, notably developments realized as part of WP3 (e.g., edge server functionalities and filtering), WP5 (e.g., security and access control), WP4 (e.g., scheduling, cloud storage) and WP6 (e.g., Github infrastructure) of the project. Note that the present deliverable presents the final release of the middleware platform of the project, which will be validated in WP6 (based on the development of the final versions of the OpenIoT use cases) and evaluated in WP7 (as part of the techno-economic evaluation task).

Due to the open source nature of the project, the middleware infrastructure described in this deliverable is publically available. OpenIoT is gradually building an open source community around this middleware infrastructure, notably a community of IoT researchers and engineers that make use of OpenIoT in order to gain insights on IoT/cloud developments, but also in order to build IoT solutions in the cloud.

OpenIoT has already received positive feedback about its middleware infrastructure, while it has also received recognition as a recipient of the Black Duck Software Co. Award “open source rookie of the year 2013 in the area of the Internet of Things”. This provides a sound basis for continuing the development of the middleware infrastructure of the project as part of OpenIoT’s sustainability phase.



OpenIoT 2014

APPENDIX I – SCHEMATA

Table 24: OSDSpec Schema

```

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

<!-- Author: Nikos Kefalakis (nkef@ait.edu.gr) -->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" targetNamespace="http://www.openiot.eu/osdspec"
  xmlns:osd="http://www.openiot.eu/osdspec"
  xmlns:prt="http://www.w3.org/2007/SPARQL/protocol-types#">

  <xs:import namespace="http://www.w3.org/2007/SPARQL/protocol-types#"
    schemaLocation="sparql/protocol-types.xsd" />

  <xs:element name="OSDSpec">
    <xs:annotation>
      <xs:documentation>OpenIoT Service Description Specification
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="osd:OAMO" />
      </xs:sequence>
      <xs:attribute name="userID" use="required" type="xs:anyURI" />
    </xs:complexType>
  </xs:element>

  <xs:element name="OAMO">
    <xs:annotation>
      <xs:documentation>OpenIoT Application Model Object
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="1" ref="osd:description" />
        <xs:element minOccurs="0" maxOccurs="1" ref="osd:graphMeta" />
        <xs:element maxOccurs="unbounded" ref="osd:OSMO" />
      </xs:sequence>
      <xs:attribute name="id" use="optional" type="xs:anyURI" />
      <xs:attribute name="name" type="xs:NCName" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="graphMeta" type="xs:string" />
  <xs:element name="description" type="xs:string" />

```

```

<xs:element name="OSMO">
  <xs:annotation>
    <xs:documentation>OpenIoT Service Model Object</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="1" ref="osd:description" />
      <xs:element ref="osd:queryControls" />
      <xs:element ref="osd:requestPresentation" />
      <xs:sequence>
        <xs:element ref="prt:query-request" maxOccurs="unbounded"
          minOccurs="1" />
      </xs:sequence>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="0"
          ref="osd:dynamicAttrMaxValue" />
      </xs:sequence>
    </xs:sequence>
    <xs:attribute name="id" use="optional" type="xs:anyURI" />
    <xs:attribute name="name" type="xs:NCName" use="required" />
  </xs:complexType>
</xs:element>

<xs:element name="serviceID">
  <xs:complexType>
    <xs:sequence>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="serviceName">
  <xs:complexType>
    <xs:sequence>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="serviceDescription">
  <xs:complexType>
    <xs:sequence>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="queryControls">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="osd:QuerySchedule" />
      <xs:element name="trigger" type="xs:anyURI"
        minOccurs="0" />
      <xs:element name="initialRecordTime" type="xs:dateTime"
        minOccurs="0" />
      <xs:element name="reportIfEmpty" type="xs:boolean" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

<xs:element name="QuerySchedule">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="second" type="xs:string" minOccurs="0" />
      <xs:element name="minute" type="xs:string" minOccurs="0" />
      <xs:element name="hour" type="xs:string" minOccurs="0" />
      <xs:element name="dayOfMonth" type="xs:string"
        minOccurs="0" />
      <xs:element name="month" type="xs:string" minOccurs="0" />
      <xs:element name="dayOfWeek" type="xs:string"
        minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="requestPresentation">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="osd:widget" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="widget">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="osd:presentationAttr" />
    </xs:sequence>
    <xs:attribute name="widgetID" use="required" type="xs:anyURI" />
  </xs:complexType>
</xs:element>

<xs:element name="presentationAttr">
  <xs:complexType>
    <xs:attribute name="name" use="required" type="xs:string" />
    <xs:attribute name="value" use="required" type="xs:string" />
  </xs:complexType>
</xs:element>

<xs:element name="dynamicAttrMaxValue">
  <xs:annotation>
    <xs:documentation>Maximum/Area of Interest for the defined dynamic
      value
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:attribute name="name" use="required" type="xs:string" />
    <xs:attribute name="value" use="required" type="xs:string" />
  </xs:complexType>
</xs:element>

</xs:schema>

```

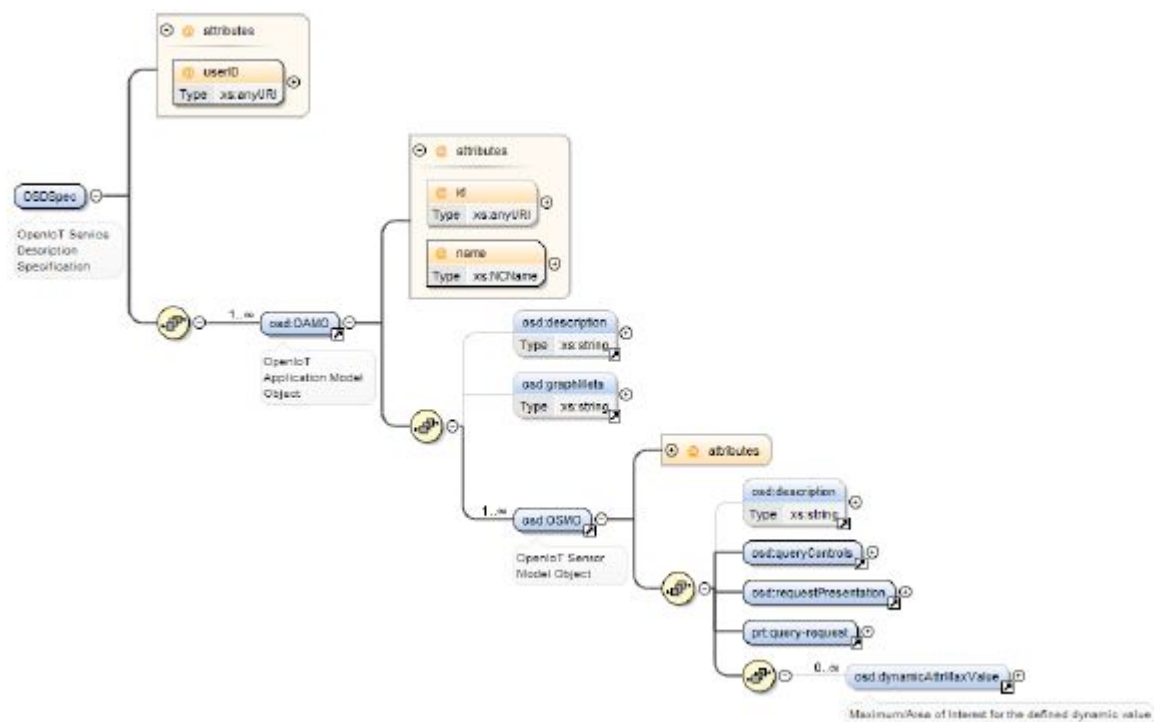


Figure 28: OSDSpec Schema graph

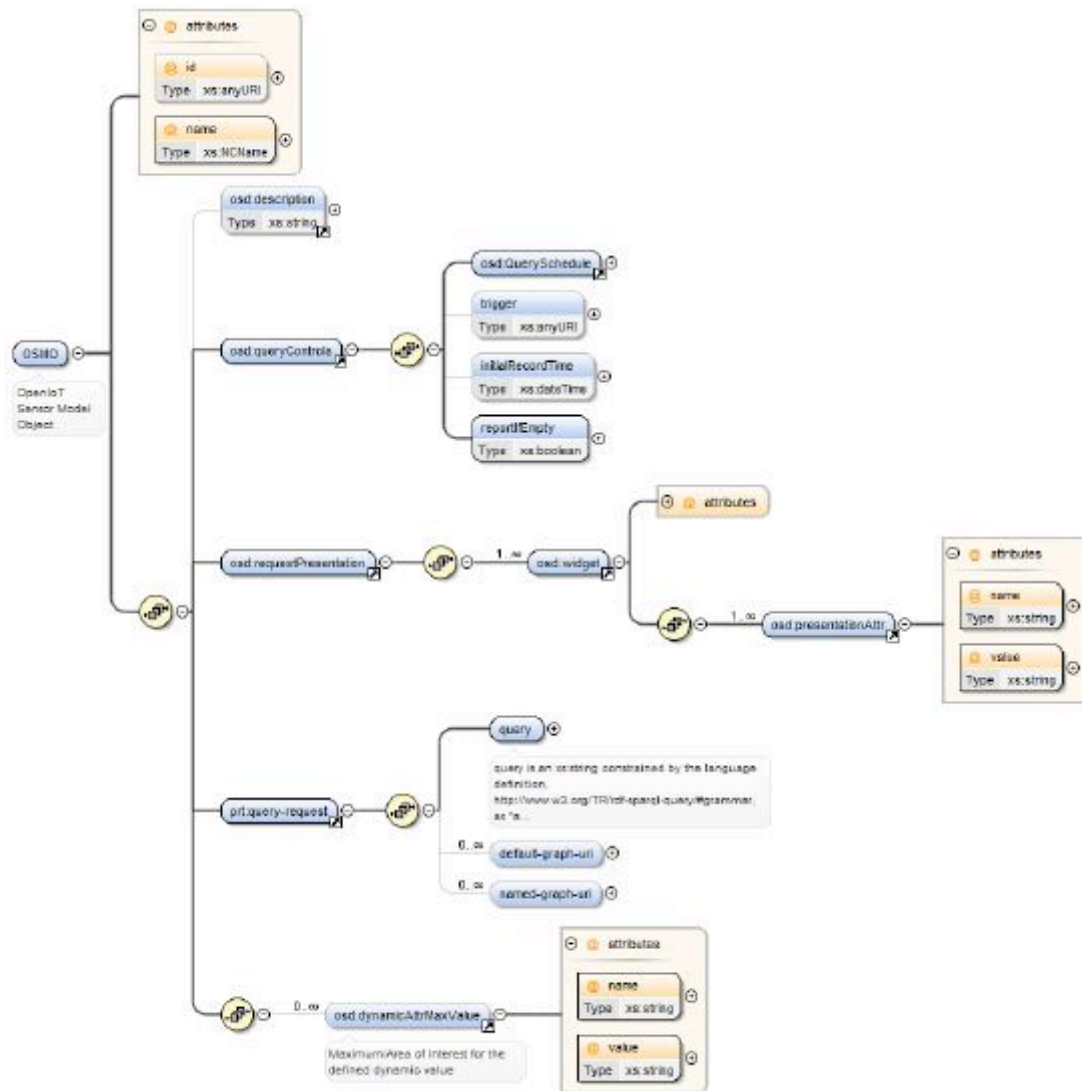


Figure 29: OSMO Schema graph

Table 25: SdumServiceResultSet Schema

```

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

<!-- Author: Nikos Kefalakis (nkef@ait.edu.gr) -->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  targetNamespace="urn:openiot:sdum:serviceresultset:xsd:1"
  xmlns:pvt="http://www.w3.org/2007/SPARQL/protocol-types#"
  xmlns:srs="urn:openiot:sdum:serviceresultset:xsd:1">

  <xs:import namespace="http://www.w3.org/2007/SPARQL/protocol-types#"
    schemaLocation="sparql/protocol-types.xsd"/>

```

```

<xs:element name="SdumServiceResultSet">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="srs:requestPresentation" maxOccurs="1"/>
      <xs:sequence>
        <xs:element ref="prt:query-result" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="requestPresentation">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="srs:widget"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="widget">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="srs:presentationAttr"/>
    </xs:sequence>
    <xs:attribute name="widgetID" use="required" type="xs:anyURI"/>
  </xs:complexType>
</xs:element>

<xs:element name="presentationAttr">
  <xs:complexType>
    <xs:attribute name="name" use="required" type="xs:string"/>
    <xs:attribute name="value" use="required" type="xs:string"/>
  </xs:complexType>
</xs:element>
</xs:schema>

```

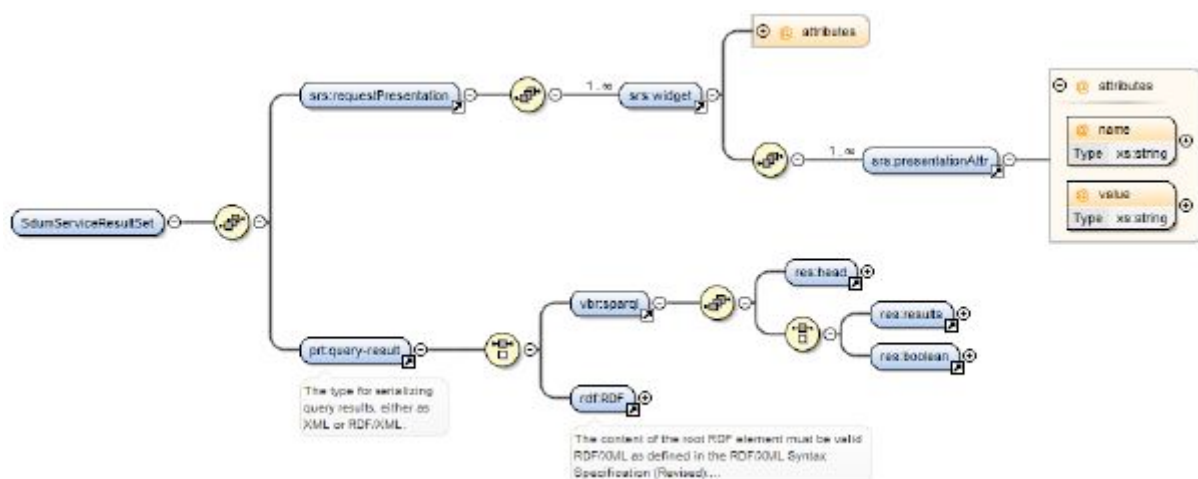


Figure 30: SdumServiceResultSet Schema graph

Table 26: SensorTypes Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <!-- Author: Nikos Kefalakis (nkef@ait.edu.gr) -->

  <xs:element name="SensorTypes">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="SensorType" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="SensorType">
    <xs:complexType>
      <xs:sequence>
        <xs:sequence>
          <xs:element maxOccurs="unbounded" minOccurs="0"
            ref="CoreMetaData" />
        </xs:sequence>
        <xs:sequence>
          <xs:element maxOccurs="unbounded" minOccurs="0"
            ref="MeasurementCapability" />
        </xs:sequence>
      </xs:sequence>
      <xs:attribute name="name" type="xs:Name" />
      <xs:attribute name="id" type="xs:anyURI" />
    </xs:complexType>
  </xs:element>

  <xs:element name="CoreMetaData" nillable="false">
    <xs:complexType>
      <xs:attribute name="name" type="xs:anyURI" />
      <xs:attribute name="value" type="xs:string" />
    </xs:complexType>
  </xs:element>

  <xs:element name="MeasurementCapability" nillable="false">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="1" ref="Unit" />
      </xs:sequence>
      <xs:attribute name="id" type="xs:anyURI" use="optional" />
      <xs:attribute fixed="" name="type" type="xs:string" />
    </xs:complexType>
  </xs:element>

```

```

<xs:element name="Unit">
  <xs:complexType>
    <xs:attribute name="name" type="xs:string" />
    <xs:attribute name="type" type="xs:string" />
  </xs:complexType>
</xs:element>

</xs:schema>

```

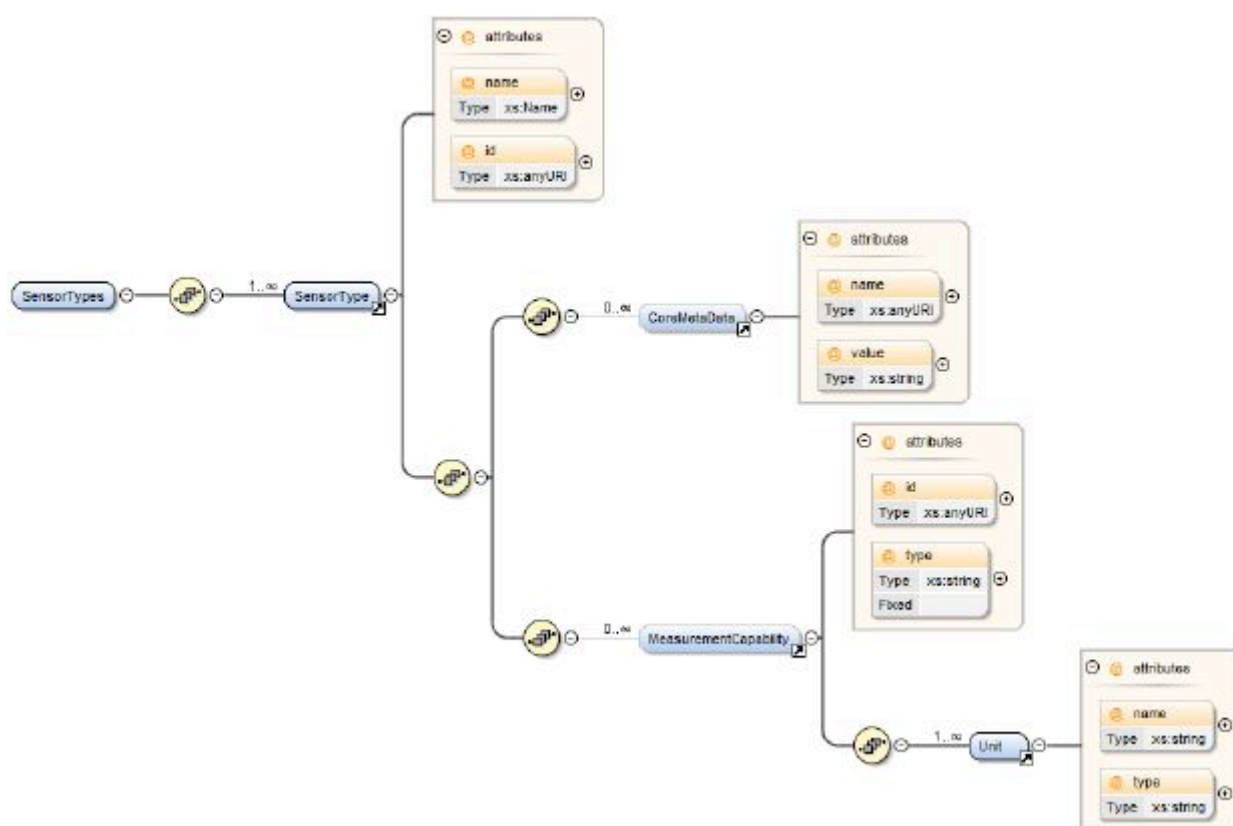


Figure 31: SensorTypes Schema graph

Table 27: DescriptiveIDs Schema

```

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

<!-- Author: Nikos Kefalakis (nkef@ait.edu.gr) -->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <xs:element name="DescriptiveIDs">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="DescriptiveID" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="DescriptiveID">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="1" name="description"
          type="xs:string" />
        <xs:element minOccurs="0" maxOccurs="1" name="name"
          type="xs:NCName" />
      </xs:sequence>
      <xs:attribute name="id" type="xs:anyURI" />
    </xs:complexType>
  </xs:element>
</xs:schema>

```

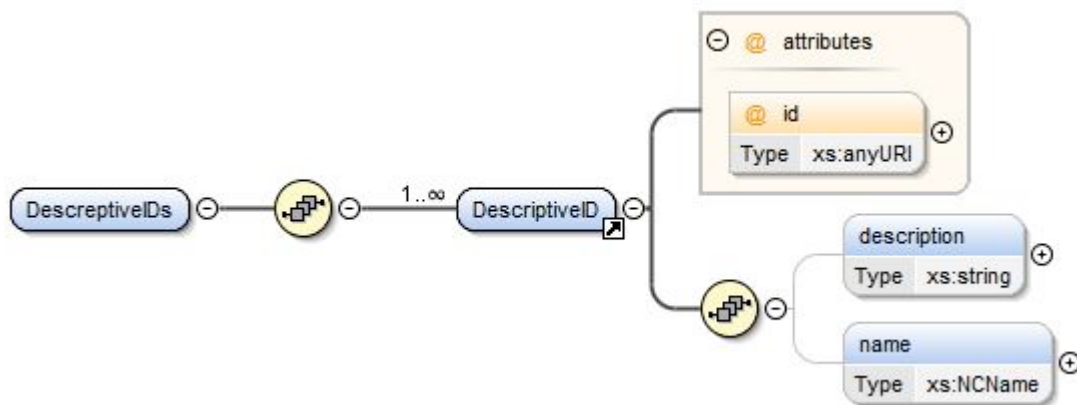


Figure 32: DescriptiveIDs Schema graph

APPENDIX II – EXTENDING X-GSN WITH NEW WRAPPERS AND VIRTUAL PROCESSING CLASSES

Writing new wrappers

All standard wrappers are subclasses of *gsn.wrapper.AbstractWrapper*. Subclasses must implement the following four (4) methods:

1. `boolean initialize()`
2. `void finalize()`
3. `String getWrapperName()`
4. `DataField[] getOutputFormat()`

Each wrapper is a thread in the X-GSN. If you want to do some kind of processing in a fixed time interval, you can override the **run()** method. The run method is useful for time driven wrappers in which the production of a sensor data is triggered by a timer. Optionally, you may wish to override the method

```
boolean sendToWrapper(String action, String[] paramNames, Object[] paramValues);
```

initialize() method

This method is called after the wrapper object creation. The complete method prototype is as follows:

```
public boolean initialize();
```

In this method, the wrapper should try to initialize its connection to the actual data producing/receiving device(s) (e.g., wireless sensor networks or cameras). The wrapper should return true if it can successfully initialize the connection, false otherwise. X-GSN provides access to the wrapper parameters through the following method call:

```
getActiveAddressBean().getPredicateValue("parameter-name");
```

For example, if you have the following fragment in the virtual sensor configuration file:

```
<source ... >
  <address wrapper="x">
    <predicate key="range">100</predicate>
    <predicate key="log">0</predicate>
  </address>
```

You can access the initialization parameter named range with the following code:

```
if(getActiveAddressBean().getPredicateValue("range") != null)
{...}
```

By default X-GSN assumes that the timestamps of the data produced in a wrapper are local, that is, the wrapper produced them using the system (or X-GSN) time. If you have cases where this assumption is not valid and X-GSN should assume remote timestamps for stream elements, add the following line in the **initialize()** method:

```
setUsingRemoteTimestamp(true);
```

finalize()method

In **finalize()** method, you should release all the resources you acquired during the initialization procedure or during the life cycle of the wrapper. Note that this is the last chance for the wrapper to release all its reserved resources and after this call the wrapper instance virtually won't exist anymore. For example, if you open a file in the initialization phase, you should close it in the finalization phase.

getWrapperName()method

This method returns a name for the wrapper.

getOutputFormat()method

The method **getOutputFormat()** returns a description of the data structure produced by this wrapper. This description is an array of DataField objects. A DataField object can be created with a call to the constructor

```
public DataField(String name, String type, String description)
```

The name is the field name, the type is one of X-GSN data types (*TINYINT*, *SMALLINT*, *INTEGER*, *BIGINT*, *CHAR*(\#), *BINARY*(\#), *VARCHAR*(\#), *DOUBLE*, *TIME*)⁴⁹, and description is a text describing the field. The following examples should help you get started:

Wireless Sensor Network Example

Assuming that you have a wrapper for a wireless sensor network which produces the average temperature and light value of the nodes in the network, you can implement ***getOutputFormat()*** as follows:

```
public DataField[] getOutputFormat() {
    DataField[] outputFormat = new DataField[2];
    outputFormat[0] = new DataField("Temperature", "double",
        "Average of temperature readings from the sensor network");
    outputFormat[1] = new DataField("light", "double",
        "Average of light readings from the sensor network");
    return outputFormat;
}
```

Webcam Example 1

If you have a wrapper producing jpeg images as output (e.g. from wireless camera), the method is similar, as shown below:

```
public DataField[] getOutputFormat() {
    DataField[] outputFormat = new DataField[1];
    outputFormat[0] = new DataField("Picture", "binary:jpeg",
        "Picture from the Camera at room BC143");
    return outputFormat;
}
```

run() method

As described before, the wrapper acts as a bridge between the actual hardware device(s) or other kinds of stream data sources and X-GSN, thus in order for the wrapper to produce data, it should keep track of the newly produced data items. This method is responsible for forwarding the newly received data to the X-GSN engine. You should not try to start the thread by yourself as X-GSN takes care of this.

⁴⁹ See ***gsn.beans.DataTypes*** package

The method should be implemented as below:

```
while(isActive()) {  
    {  
        // The thread should wait here until arrival of a new data notifies it  
        // or the thread should sleep for some finite time before polling the data  
        source or producing the next data  
    }  
  
    //Application dependent processing ...  
  
    StreamElement streamElement = new StreamElement ( ...);  
  
    postStreamElement( streamElement ); // This method in the AbstractWrapper sends  
    the data to the registered StreamSources  
}
```

Webcam Example 2

Assume that we have a wireless camera which runs a HTTP server and provides pictures whenever it receives a **GET** request. In this case we are in a data on demand scenario (most of the network cameras are like this). To get the data at the rate of 1 picture every 5 seconds we can do the following:

```
while(isActive()) {  
    byte[] received_image = getPictureFromCamera();  
    postStreamElement(System.currentTimeMillis(), new Serializable[]  
{received_image});  
  
    Thread.sleep(5*1000); // Sleeping 5 seconds  
}
```

Data driven systems

Compared to the previous example, we do sometimes deal with devices that are data driven. This means that we don't have control of either when the data is produced by them (e.g., when they do the capturing) or the rate at which data is received from them. For example, having an alarm system, we don't know when we are going to receive a packet, or how

frequently the alarm system will send data packets to X-GSN. These kinds of systems are typically implemented using a callback interface. In the callback interface, one needs to set a flag indicating the data reception state of the wrapper and control that flag in the run method to process the received data.

sendToWrapper()

In X-GSN, the wrappers can not only receive data from a source, but also send data to it. Thus wrappers are actually two-way bridges between X-GSN and the data source. In the wrapper interface, the method **sendToWrapper()** is called whenever there is a data item which should be send to the source. A data item could be as simple as a command for turning on a sensor inside the sensor network, or it could be as complex as a complete routing table which should be used for routing the packets in the sensor network. The full syntax of **sendToWrapper()** is as follows:

```
public boolean sendToWrapper(String action, String[] paramNames, Object[] paramValues) throws UnsupportedOperationException;
```

The default implementation of the afore-mentioned method throws an **OperationNotSupportedException** exception because the wrapper doesn't support this operation. This design choice is justified by the observation that not all kind of devices (sensors) can accept data from a computer. For instance, a typical wireless camera doesn't accept commands from the wrapper. If the sensing device supports this operation, one needs to override this method so that instead of the default action (throwing the exception), the wrapper sends the data to the sensor. You can consult the `gsn.wrappers.general.SerialWrapper` class for an example.

A fully functional wrapper

This wrapper presents a MultiFormat protocol in which the data comes from the system clock. Think about a sensor network which produces packets with several different formats. In this example we have 3 different packets produced by three different types of sensors. Here are the packet structures: [temperature:double], [light:double] and [temperature:double, light:double]. The first packet is for sensors which can only measure temperature while the latter is for the sensors equipped with both temperature and light sensors.

```
public class MultiFormatWrapper extends AbstractWrapper {  
    private DataField[] collection = new DataField[] { new DataField("packet_type",  
        "int", "packet type"),  
        new DataField("temperature", "double", "Presents the temperature sensor."),  
        new DataField("light", "double", "Presents the light sensor.") };  
    private final transient Logger logger =  
        Logger.getLogger(MultiFormatWrapper.class);
```

```
private int counter;
private AddressBean params;
private long rate = 1000;

public boolean initialize() {
    setName("MultiFormatWrapper" + counter++);

    params = getActiveAddressBean();

    if ( params.getPredicateValue( "rate" ) != null ) {
        rate = (long) Integer.parseInt( params.getPredicateValue( "rate" ));

        logger.info("Sampling rate set to " + params.getPredicateValue( "rate" ) + "
msec.");
    }

    return true;
}

public void run() {
    Double light = 0.0, temperature = 0.0;
    int packetType = 0;

    while (isActive()) {
        try {

            // delay

            Thread.sleep(rate);
        } catch (InterruptedException e) {
            logger.error(e.getMessage(), e);
        }

        // create some random readings

        light = ((int) (Math.random() * 10000)) / 10.0;
        temperature = ((int) (Math.random() * 1000)) / 10.0;
        packetType = 2;

        // post the data to GSN

        postStreamElement(new Serializable[] { packetType, temperature, light });
    }
}
```

```
public DataField[] getOutputFormat() {  
    return collection;  
}  
  
public String getWrapperName() {  
    return "MultiFormat Sample Wrapper";  
}  
  
public void finalize() {  
    counter--;  
}  
}
```

Writing new processing classes

In GSN, a processing class is a piece of code which acts in the final stage of data processing as it sits between the wrapper and the data publishing engine. The processing class is the last processing stage on the data and its inputs are specified in the virtual sensor file.

All virtual sensors are subclass of the **AbstractVirtualSensor** (package *gsn.vsensor*). It requires its subclasses to implement the following three methods:

```
public boolean initialize();  
public void dataAvailable(String inputStreamName, StreamElement se);  
public void dispose();
```

initialize() method

initialize is the first method to be called after object creation. This method should configure the virtual sensor according to its parameters, if any, and return true in case of success, false if otherwise. If this method returns false, X-GSN will generate an error message in the log and stops using the processing class hence stops loading the virtual sensor.

Dispose() method

dispose is called when X-GSN destroys the virtual sensor. It should release all system resources in use by this virtual sensor. This method is typically called when we want to shut down the X-GSN instance.

dataAvailable() method

dataAvailable is called each time that X-GSN has data for this processing class, according to the virtual sensor file. If the processing class produces data, it should encapsulate this data in a **StreamElement** object and deliver it to GSN by calling **dataProduced(StreamElement se)** method.

Note that a processing class should always use the same **StreamElement** structure for delivering its output. Changing the structure type is not allowed and trying to do so will result in an error. However, a virtual sensor can be configured at initialization time with the kind of **StreamElement** it will produce (e.g. setting the output type to be the super set of all the possible outputs and providing null whenever the value is missing).

This allows producing different types of **StreamElements** by the same VS depending on its usage. But one instance of the VS will still be limited to produce the same structure type. If a virtual sensor really needs to produce several different stream elements, user must provide the set of all possibilities in the stream elements and provide Null whenever the data item is not applicable.

The processing class can read the **init** parameters from the virtual sensor description file. Example:

```
<class-name>gsn.vsensor.MyProcessor</class-name>
  <init-params>
    <param name="param1">DATA</param>
    <param name="param2">1234</param>
  </init-params>
```

And inside the processing class's initialization method:

```
String param1 = getVirtualSensorConfiguration().getMainClassInitialParams().get (
"param1" );
String param2 = getVirtualSensorConfiguration().getMainClassInitialParams().get (
"param2" );
```

APPENDIX III – OPENIOT INSTALLATION QUICK GUIDE

At OpenIoT Project, most of the components of the implemented OpenIoT middleware are written in Java programming language, to implement the OpenIoT platform correctly this section describes a method that has been tested, you can use any of the linux distribution and follow the pre-requisites that are advised as follow.

1. JDK 1.7
2. Maven 3.0
3. JBoss AS 7
4. (Optional) Eclipse or any other IDE that you are familiar with
5. The source code

If you do not have already installed any of the unix distribution, you can do it from:
<http://www.ubuntu.com/download/desktop/install-ubuntu-with-windows>

We will guide you through the prerequisite section. Remember this is just a guideline; you can follow your own procedure to acquire them.

Prerequisite 1: JDK 1.7

You can use this link for help:

><http://askubuntu.com/questions/56104/how-can-i-install-sun-oracles-proprietary-java-6-7-jre-or-jdk>

Note: We recommend you to install Sun/Oracle Java, not the OpenJDK.
check if u have done it correctly by typing in terminal:
`java -version`

Prerequisite 2: Maven 3.0

For installing maven on ubuntu, follow this procedure:

<http://stackoverflow.com/questions/15630055/how-to-install-maven-3-on-ubuntu-12-04-12-10-13-04-13-10-by-using-apt-get>

Alternatively you can try installing maven from Eclipse as well:

<http://askubuntu.com/questions/141204/what-is-the-correct-way-to-install-maven-and-eclipse>

Check the maven version:
`mvn -version`

Prerequisite 3: JBoss AS 7

For installing JBoss on ubuntu, you can follow these two tutorials, and do not need to create a new user thus do not follow the step three of the first link (Start from step 2, as it is expected that u have installed java 7 beforehand).

><https://www.digitalocean.com/community/articles/how-to-install-jboss-on-ubuntu-12-10-64bit>

><http://dont-panic.eu/blogs/2012/mar/install-jboss-7-ubuntu-1110-server>

After Installation,

Start the JBoss Enterprise Application Platform 6 or JBoss AS 7.1 with the Web Profile.

Open a command line and navigate to the root of the JBoss server directory.

The following shows the command line to start the server with the web profile:

For Linux: `JBOSS_HOME/bin/standalone.sh`

For Windows: `JBOSS_HOME\bin\standalone.bat`

For example, we have used: `XXX@ubuntu:~/jboss-as-7.1.1.Final/bin$./standalone.sh`

Should get results here (<http://localhost:8080/>)

Alternatively you can also try to install JBoss from Eclipse Marketplace:

If there is no 'marketplace' for your ubuntu, install so:

><http://stackoverflow.com/questions/11403147/install-marketplace-plugin-on-eclipse-juno>

Prerequisite 4. (Optional) Install Eclipse environment or any other IDE that you are familiar and feel confident to edit visualise java code.

Prerequisite 5: Getting the Binaries (Java Code)

To deploy the codes, you have to acquire the codes first. For that you can 'install git client' by executing the following command:

```
sudo apt-get install git
```

And *get the codes* using following command:

```
git clone https://github.com/OpenIoTOrg/openiot.git
```

With this command all the codes supposed to be in `~/openiot/`

Now unzip the openiot-modules.

Now you have the prerequisites ready and you have the codes in your local machine. OpenIoT modules depends on each other, so it is recommended that you follow the instructed serial while deploying the full platform.

Please note that several links with instructions suggested below may request and instruct you to install the prerequisites (e.g: Java or maven) again. Ignore those whenever necessary.

During the deployment steps, occasionally, your build may fail. It can happen for several reasons. If unsuccessful at the first attempt, try multiple times to deploy the same module. If it doesn't work, please swap between link (1) and (2). For example in Deployment Step 2, we have provided you two links. If the mentioned serial doesn't work, try the opposite - first follow link 2, then link 1. By doing this you should be able to deploy/install the module.

Deployment Step 1

utils.common

Follow this link: <https://github.com/OpenlotOrg/openiot/wiki/Openiot-Commons-Library>

copy utils.common file from (example location:

~/openiot/utils/utils.common/src/main/resources/properties/) to jboss directory (example location: ~/jboss-as-7.1.1.Final/standalone/configuration/)

Some properties of this properties file need to be modified depending on the graph you use or data you load into your database. We will discuss about it later in this document.

Deployment Step 2

Lsm-light client and server

Follow these links:

(1) <https://github.com/OpenlotOrg/openiot/wiki/Sensor-Data-Use>

(2) <https://github.com/OpenlotOrg/openiot/wiki/Sensor-Data-Develop>

Deployment Step 3

Scheduler

Follow these links:

(1) <https://github.com/OpenlotOrg/openiot/wiki/Global-Scheduler-Use>

(2) <https://github.com/OpenlotOrg/openiot/wiki/Global-Scheduler-Develop>

Check: <http://localhost:8080/scheduler.core/rest/services>

Deployment Step 4

SDUM

Follow these links:

(1) <https://github.com/OpenlotOrg/openiot/wiki/SD%26UM-Use>

(2) <https://github.com/OpenlotOrg/openiot/wiki/SD%26UM-Develop>

Check: <http://localhost:8080/sdum.core/rest/services>

Deployment Step 5

request.common

Go to ui/ui.requestCommons directory and install the ui.requestCommons.

Deployment Step 6

ide.core

Follow the link:

(1) <https://github.com/OpenlotOrg/openiot/wiki/IDE-Core-Use>

(2) <https://github.com/OpenlotOrg/openiot/wiki/IDE-Core-Develop>

Check: <http://localhost:8080/ide.core/>

Deployment Step 7

request definition

Follow the links:

- (1) <https://github.com/OpenIoTOrg/openiot/wiki/Request-definition-use>
 - (2) <https://github.com/OpenIoTOrg/openiot/wiki/Request-Definition-Develop>
- Check: <http://localhost:8080/ui.requestDefinition/>

Deployment Step 8

request presentation

Follow the links:

- (1) <https://github.com/OpenIoTOrg/openiot/wiki/Request-presentation-use>
 - (2) <https://github.com/OpenIoTOrg/openiot/wiki/Request-Presentation-Develop>
- Check: <http://localhost:8080/ui.requestPresentation/>

Deployment Step 9

Visual Sensor Schema Editor - RDFSensorSchemaEditor

Follow the links:

- (1) <https://github.com/OpenIoTOrg/openiot/wiki/Virtual-Sensor-RDF-Schema-Editor-Use>
- (2) <https://github.com/OpenIoTOrg/openiot/wiki/Virtual-Sensor-RDF-Schema-Editor-Develop>

Check: <http://localhost:8080/sensorschema>

By now, your openiot platform should be ready to run. But you do not have any data to test your platform. To load data to virtuoso database, firstly you have to have the data, secondly you have to load the data into the database. You can follow this step for loading data:

<http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtBulkRDFLoader>

Your openiot.properties file placed in JBoss directory must be configured according to your loaded graph, data and other parameters. It is important for you to understand what kind of graph you are loading for what purpose. For example, we have loaded these following graphs to our virtuoso database, and thus we have used the relevant openiot.properties file attached:

graphs loaded:

<http://lsm.deri.ie/OpenIoT/demo/sensordata#>
<http://lsm.deri.ie/OpenIoT/demo/functionaldata#>
<http://lsm.deri.ie/OpenIoT/demo/sensormeta#>

Openiot.properties file:

**** its better to provide a link of the openiot.properties file here ****

#Scheduler Properties

#The following three graph names depend on the graphs you imported in your DB
scheduler.core.lsm.openiotMetaGraph=<http://lsm.deri.ie/OpenIoT/demo/sensormeta#>

```
scheduler.core.lsm.openiotDataGraph=http://lsm.deri.ie/OpenIoT/demo/sensordata#
scheduler.core.lsm.openiotFunctionalGraph=http://lsm.deri.ie/OpenIoT/demo/functionald
ata#
```

#At this moment, you do not need to change the credentials below

```
scheduler.core.lsm.access.username=openiot_guest
scheduler.core.lsm.access.password=openiot
```

#Next two parameters are configured for OpenIoT using local database

```
scheduler.core.lsm.sparql.endpoint=http://localhost:8890/sparql
scheduler.core.lsm.remote.server=http://localhost:8080/lsm-light.server/
```

#Service Delivery & Utility Manager (SD&UM) Properties

#The following graph name depends on the graph you imported in your DB

```
sdum.core.lsm.openiotFunctionalGraph=http://lsm.deri.ie/OpenIoT/demo/functionaldata#
```

#Next two parameters are configured for OpenIoT using local database

```
sdum.core.lsm.sparql.endpoint=http://localhost:8890/sparql
sdum.core.lsm.remote.server=http://localhost:8080/lsm-light.server/
```

#Request Definition

#Request Presentation

#LSM-LIGHT Properties

```
lsm-light.server.connection.driver_class=virtuoso.jdbc4.Driver
lsm-light.server.connection.url=jdbc:virtuoso://localhost:1111/log_enable=2
```

#The following credentials depend on your installation of local DB (virtuoso)

```
lsm-light.server.connection.username=dba
lsm-light.server.connection.password=dba
```

```
lsm-light.server.minConnection=10
lsm-light.server.maxConnection=15
lsm-light.server.acquireRetryAttempts=5
```

#for local virtuoso instance

#The following graph names depend on the graphs you imported in your DB

```
lsm-light.server.localMetaGraph = http://lsm.deri.ie/OpenIoT/demo/sensormeta#
lsm-light.server.localDataGraph = http://lsm.deri.ie/OpenIoT/demo/sensordata#
-----
```

By following this procedure you should be able to have up and running an OpenIoT instance with all the core functionality enabled and ready to use it.