D4.3.1 Core OpenIoT Middleware Platform

Document Id: OpenIoT-D431-130927-Draft
File Name: OpenIoT-D431-130927-Draft.pdf
Document reference: Deliverable 4.3.1
Version: Draft
Editor(s): Nikos Kefalakis, Stavros Petris, John Soldatos
Organisation: AIT
Date: 2013/09/27
Document type: Deliverable (Prototype)
Dissemination level: PU (Public)

Copyright © 2013 OpenIoT Consortium: NUIG-National University of Ireland Galway, Ireland; EPFL - École 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; UniZ-FER University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia. Project co-funded by the European Commission within FP7 Program.
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Author(s)</th>
<th>Organisation(s)</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>V01</td>
<td>Nikos Kefalakis, John Soldatos</td>
<td>AIT</td>
<td>2013/05/30</td>
<td>ToC</td>
</tr>
<tr>
<td>V02</td>
<td>Nikos Kefalakis</td>
<td>AIT</td>
<td>2013/06/28</td>
<td>Added “Download, Deploy &amp; Run of Scheduler and SD&amp;UM,” “Source code availability and Structure”, “Platform containers dev tools and libraries” Scheduler and SD&amp;UM libraries used.</td>
</tr>
<tr>
<td>V03</td>
<td>John Soldatos</td>
<td>AIT</td>
<td>2013/07/22</td>
<td>Introduction</td>
</tr>
<tr>
<td>V04</td>
<td>Sofiane Sarni</td>
<td>EPFL</td>
<td>2013/07/24</td>
<td>Added content of X-GSN</td>
</tr>
<tr>
<td>V05</td>
<td>Hoan Nguyen Mau Quoc</td>
<td>DERI</td>
<td>2013/07/24</td>
<td>Added LSM-Light System Requirements &amp; Libraries</td>
</tr>
<tr>
<td>V06</td>
<td>Nikos Kefalakis</td>
<td>AIT</td>
<td>2013/07/28</td>
<td>Corrections, Merging of Versions, Overview of Architecture, Scheduler and SD&amp;UM Services/Implementation description and usage, Various Editing</td>
</tr>
<tr>
<td>V07</td>
<td>Aquileas Anagnostopoulos Panos Dimitropoulos</td>
<td>SENSAP</td>
<td>2013/09/04</td>
<td>Inputs and Documentation on section 3.5 User Interfaces</td>
</tr>
<tr>
<td>V08</td>
<td>Sofiane Sarni</td>
<td>EPFL</td>
<td>2013/09/05</td>
<td>Update of X-GSN content</td>
</tr>
<tr>
<td>V09</td>
<td>Hoan Nguyen Mau Quoc</td>
<td>DERI</td>
<td>2013/09/09</td>
<td>Updated LSM-Light content</td>
</tr>
<tr>
<td>V10</td>
<td>John Soldatos</td>
<td>AIT</td>
<td>2013/09/10</td>
<td>Conclusion Section</td>
</tr>
<tr>
<td>V12</td>
<td>Ivana Podnar Aleksandar Antonić</td>
<td>FER</td>
<td>2013/09/13</td>
<td>TR. Comments provided.</td>
</tr>
<tr>
<td>V13</td>
<td>Nikos Kefalakis</td>
<td>AIT</td>
<td>2013/09/13</td>
<td>Comments from TR addressed.</td>
</tr>
<tr>
<td>V14</td>
<td>Hoan Nguyen Mau Quoc</td>
<td>DERI</td>
<td>2013/09/16</td>
<td>Comments from TR addressed for Section 3.3 (LSM-Light). Document ready for QR.</td>
</tr>
<tr>
<td>V15</td>
<td>Tian Guo</td>
<td>EPFL</td>
<td>2013/09/18</td>
<td>QR Comments provided.</td>
</tr>
<tr>
<td>V17</td>
<td>Martin Serrano</td>
<td>DERI</td>
<td>2013/09/23</td>
<td>Circulation for Approval</td>
</tr>
<tr>
<td>V18</td>
<td>Martin Serrano</td>
<td>DERI</td>
<td>2013/09/25</td>
<td>Approved</td>
</tr>
<tr>
<td>Draft</td>
<td>Martin Serrano</td>
<td>DERI</td>
<td>2013/09/27</td>
<td>EC Submitted</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

1 INTRODUCTION ......................................................... 10
   1.1 SCOPE ...................................................................... 10
   1.2 AUDIENCE .................................................................. 11
   1.3 SUMMARY .................................................................. 12
   1.4 STRUCTURE ................................................................ 12

2 OVERVIEW OF OPEN SOURCE RELEASE ARCHITECTURE .......... 13
   2.1 OVERVIEW OF OPENIoT ARCHITECTURE ...................... 13
   2.2 OPENIoT ARCHITECTURE PROOF OF CONCEPT ............. 15
       2.2.1 Data Flow .......................................................... 17

3 OPENIoT COMPONENTS DEVELOP/DEPLOY/USE ..................... 19
   3.1 SCHEDULER ............................................................. 19
       3.1.1 Main Released Functionalities & Services .............. 19
       3.1.2 Download, Deploy & Run .................................... 21
           3.1.2.1 Developer .................................................. 21
           3.1.2.2 User ......................................................... 24
   3.2 SERVICE DELIVERY & UTILITY MANAGER ................... 26
       3.2.1 Main Released Functionalities & Services .............. 26
       3.2.2 Download, Deploy & Run .................................... 27
           3.2.2.1 Developer .................................................. 27
           3.2.2.2 User ......................................................... 30
   3.3 LINKED STREAM MIDDLEWARE LIGHT (LSM-LIGHT) AS THE DATA PLATFORM ........ 32
       3.3.1 Main Released Functionalities & Services .............. 32
       3.3.2 Download, Deploy and Run .................................. 35
           3.3.2.1 Developer .................................................. 35
           3.3.2.2 User ......................................................... 43
   3.4 X-GSN (EXTENDED GLOBAL SENSOR NETWORK) .......... 44
       3.4.1 Main Released Functionalities & Services .............. 44
       3.4.2 Download, Deploy and Run .................................. 45
           3.4.2.1 Developer .................................................. 45
           3.4.2.2 User ......................................................... 46
   3.5 USER INTERFACES ..................................................... 48
       3.5.1 Request Definition ............................................... 48
           3.5.1.1 Main Released Functionalities & Services .......... 49
           3.5.1.2 Reset or refresh applications ......................... 50
D4.3.1 Core OpenIoT Middleware Platform

3.5.1.3 Creating a new application ................................................................. 50
3.5.1.4 Editing an existing application ....................................................... 50
3.5.1.5 Modelling the service graph of an application ............................... 50
3.5.1.6 Using variable property values ....................................................... 57
3.5.1.7 Workspace validation ................................................................. 58
3.5.1.8 Deploy from source .................................................................... 58
3.5.1.9 Developers Guide ..................................................................... 60
3.5.2 Request Presentation ...................................................................... 61
3.5.2.1 Deploy From Source ................................................................... 62
3.5.2.2 Developers Guide ..................................................................... 63

4 SOURCE CODE & BINARIES .................................................................. 64
4.1 SOURCE CODE AVAILABILITY ....................................................... 64
4.2 SOURCE CODE STRUCTURE .............................................................. 64
4.3 BINARIES AVAILABILITY .................................................................. 65

5 PLATFORM CONTAINERS, DEVELOPMENT TOOLS & LIBRARIES USED ... 66
5.1 OPENIoT COMPONENTS-LIBRARIES RELATION ................................. 68
5.1.1 Scheduler .................................................................................. 68
5.1.2 Service Delivery & Utility Manager ................................................ 69
5.1.3 X-GSN ..................................................................................... 69
5.1.4 LSM-Light ................................................................................. 70
5.1.5 User Interface ........................................................................... 71
5.1.5.1 Request Presentation .............................................................. 71
5.1.5.2 Request Definition ................................................................. 72

6 CONCLUSIONS .................................................................................. 73

APPENDIX I – SCHEMAT A ...................................................................... 74
APPENDIX II – EXTENDING X-GSN WITH NEW WRAPPERS AND VIRTUAL
PROCESSING CLASSES ........................................................................... 83
Writing new wrappers ............................................................................. 83
initialize() method ............................................................................. 83
finalize() method ............................................................................... 84
getWrapperName() method ................................................................. 84
getOutputFormat() method ................................................................. 84
run() method ................................................................................... 85
sendToWrapper() .............................................................................. 86
A fully functional wrapper ................................................................. 87
Writing new processing classes .............................................................. 89
initialize() method ............................................................................. 89
Dispose() method ............................................................................. 89
dataAvailable() method ..................................................................... 89
LIST OF FIGURES

FIGURE 1. OPENIoT MAIN CORE COMPONENTS FUNCTIONAL BLOCKS ........................................... 13
FIGURE 2. OPENIoT ARCHITECTURE OVERVIEW (FUNCTIONAL VIEW) ...................................... 15
FIGURE 3. OPENIoT PoC MODULES AND ENTITIES .......................................................... 16
FIGURE 4. PROOF-OF-CONCEPT ARCHITECTURE ................................................................ 18
FIGURE 5. ECLIPSE JUNO - NEW SERVER WIZARD .............................................................. 38
FIGURE 6. ECLIPSE JUNO – CHOOSING/REGISTERING YOUR SERVER .................................. 38
FIGURE 7. ECLIPSE JUNO – SETTING YOUR SERVER PATH ....................................................... 39
FIGURE 8. ECLIPSE JUNO – SELECTING YOUR NOW REGISTERED SERVER ......................... 39
FIGURE 9. ECLIPSE JUNO – REGISTERED SERVER RUNNING ............................................... 39
FIGURE 10. DATA ACQUISITION NETWORK IN X-GSN ............................................................ 44
FIGURE 11. REQUEST DEFINITION USER INTERFACE (UI) ..................................................... 49
FIGURE 12. THE SENSOR DISCOVERY DIALOG ....................................................................... 51
FIGURE 13. A SENSOR NODE AND ITS PROPERTIES ............................................................... 52
FIGURE 14. A EXAMPLE OF RELATIVE TIME FILTERING ......................................................... 53
FIGURE 15. GROUPING OPTIONS DIALOG ................................................................................ 54
FIGURE 16. GROUPING AN ATTRIBUTE BY RECORD TIME ....................................................... 54
FIGURE 17. A SERVICE THAT PLOTS THE MAX TEMPERATURE PER DAY ........................... 56
FIGURE 18. VARIABLE EDITOR DIALOG ................................................................................... 58
FIGURE 19. VARIABLE SENSOR PROPERTY EXAMPLE .......................................................... 58
FIGURE 20. REQUEST PRESENTATION UI .................................................................................. 62
FIGURE 21. OSDSpec SCHEMA GRAPH .................................................................................... 77
FIGURE 22. OSMO SCHEMA GRAPH ....................................................................................... 78
FIGURE 23. SdumSERVICEResultSET SCHEMA GRAPH ......................................................... 79
FIGURE 24. SENSORTypes SCHEMA GRAPH ............................................................................ 81
FIGURE 25. DESCRIPTIVEIDS SCHEMA GRAPH ...................................................................... 82
LIST OF TABLES

TABLE 1. LIST OF PRIMITIVES COMPRISING THE IMPLEMENTED SCHEDULER API ............ 19
TABLE 2. IMPLEMENTED SCHEDULER API DEFINITION ............................................. 20
TABLE 3. LIST OF PRIMITIVES COMPRISING THE OPENIoT SD&UM IMPLEMENTED API ..... 26
TABLE 4. SERVICE DELIVERY & UTILITY MANAGER IMPLEMENTED API DEFINITION .......... 26
TABLE 5. LIST OF PRIMITIVES COMPRISING THE OPENIoT LSM-LIGHT API ................. 32
TABLE 6. LSM-LIGHT API SPECIFICATION .................................................................. 32
TABLE 7. X-GSN MAIN FUNCTIONALITIES TO RUN FROM COMMAND LINE ............... 46
TABLE 8. SAMPLE METADATA FILE ......................................................................... 47
TABLE 9. SCHEDULER LIBRARIES AND CONTAINERS .............................................. 68
TABLE 10. SD&UM’S LIBRARIES AND CONTAINERS .................................................. 69
TABLE 11. X-GSN’S LIBRARIES AND CONTAINERS ................................................... 69
TABLE 12. LSM’S LIGHT LIBRARIES AND CONTAINERS .......................................... 70
TABLE 13. REQUEST PRESENTATION’S LIBRARIES AND CONTAINERS ..................... 71
TABLE 14. REQUEST DEFINITION’S LIBRARIES AND CONTAINERS ......................... 72
TABLE 15. OSDSPEC SCHEMA ................................................................................. 74
TABLE 16. SDUMSERVICERESULTSET SCHEMA .................................................... 78
TABLE 17. SENSORTYPES SCHEMA ......................................................................... 80
TABLE 18. DESCRIPTIVEIDS SCHEMA ..................................................................... 81
**TERMS AND ACRONYMS**

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AS</td>
<td>Application Server</td>
</tr>
<tr>
<td>DoW</td>
<td>Description-of-Work</td>
</tr>
<tr>
<td>GSN</td>
<td>Global Sensor Network</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTMLDSO</td>
<td>HyperText Markup Language Decision Support Ontology</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>ICOs</td>
<td>Internet-Connected Objects</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IERC</td>
<td>IoT European Research Cluster</td>
</tr>
<tr>
<td>IoTHTML</td>
<td>Internet of Things HyperText Markup Language</td>
</tr>
<tr>
<td>JAR</td>
<td>Java ARchive</td>
</tr>
<tr>
<td>JAX-WS</td>
<td>Java API for XML Web Services</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>JMX</td>
<td>Java Management Extensions</td>
</tr>
<tr>
<td>JSF</td>
<td>Java Server Faces</td>
</tr>
<tr>
<td>LGPLHTTP</td>
<td>Lesser General Public License Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>LSM</td>
<td>Linked Sensor Middleware</td>
</tr>
<tr>
<td>LSM-Light</td>
<td>Linked Stream Middleware Light</td>
</tr>
<tr>
<td>OAMO</td>
<td>OpenIoT Application Model Object</td>
</tr>
<tr>
<td>ORDBMS</td>
<td>Object Relational Database Management System</td>
</tr>
<tr>
<td>OSDSpec</td>
<td>OpenIoT Service Description Specification</td>
</tr>
<tr>
<td>OSGi</td>
<td>Open Service Gateway Initiative</td>
</tr>
<tr>
<td>OSMO</td>
<td>OpenIoT Service Model Object</td>
</tr>
<tr>
<td>PoC</td>
<td>Proof of Concept</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational Database Management System</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Format</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>SD&amp;UM</td>
<td>Service Delivery &amp; Utility Manager</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SPARQL</td>
<td>SPARQL Protocol and RDF Query Language</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SSN</td>
<td>Semantic Sensor Networks</td>
</tr>
<tr>
<td>SSN</td>
<td>Semantic Sensor Networks</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>WAR</td>
<td>Web application ARchive</td>
</tr>
<tr>
<td>WP</td>
<td>Work Package</td>
</tr>
<tr>
<td>WSN</td>
<td>Wireless Sensor Networks</td>
</tr>
<tr>
<td>WYSIWYG</td>
<td>What You See Is What You Get</td>
</tr>
<tr>
<td>X-GSN</td>
<td>eXtended Global Sensor Network</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
## GLOSSARY AND TERMINOLOGY

<table>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(OpenIoT) Architecture</td>
<td>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.</td>
</tr>
<tr>
<td>(OpenIoT) Middleware</td>
<td>System level software (compliant to the OpenIoT architecture), which facilitates the integration of on-demand cloud-based IoT services.</td>
</tr>
<tr>
<td>(OpenIoT) Platform</td>
<td>A set of middleware libraries and tools, which enable the development and deployment of (OpenIoT compliant) cloud-based IoT services.</td>
</tr>
<tr>
<td>(OpenIoT) Use Case</td>
<td>A goal or application serving needs of end users, which is implemented based on the OpenIoT platform.</td>
</tr>
<tr>
<td>(OpenIoT) Service</td>
<td>An IoT service deployed over the OpenIoT platform.</td>
</tr>
<tr>
<td>(OpenIoT) Scenario</td>
<td>A specific set of interactions between OpenIoT components serving the needs of an application.</td>
</tr>
<tr>
<td>(OpenIoT) Cloud</td>
<td>A set of computing resources enabling the delivery of IoT services over the network and based on the use of OpenIoT platform.</td>
</tr>
<tr>
<td>Global Scheduler</td>
<td>A software component that regulates how IoT services access the different resources managed by the OpenIoT platform.</td>
</tr>
<tr>
<td>Utility Metrics</td>
<td>A set of measures contributing to the overall value and performance of IoT services</td>
</tr>
<tr>
<td>(OpenIoT) Service Delivery</td>
<td>The process of deploying and offering an OpenIoT service, after selecting the resources that are involved in the service.</td>
</tr>
<tr>
<td><strong>(OpenIoT)</strong> Request Presentation</td>
<td>Software components that visualize the outcomes of an OpenIoT service based on the use of appropriate mashups and mashup libraries.</td>
</tr>
<tr>
<td><strong>Sensor Selection</strong></td>
<td>The process of selecting sensors that can contribute information to a particular service.</td>
</tr>
<tr>
<td><strong>Virtual Sensor</strong></td>
<td>All the physical or virtual items (i.e. services, persons, sensors, GSN nodes) which provide their information through a GSN endpoint.</td>
</tr>
<tr>
<td><strong>Sensor Discovery</strong></td>
<td>The process of detecting physical and virtual sensors, as well as of the services offered by them.</td>
</tr>
<tr>
<td><strong>Resource Discovery</strong></td>
<td>The process of detecting an IoT resource (such as a sensor, service or database).</td>
</tr>
<tr>
<td><strong>Sensor Directory</strong></td>
<td>A software service which stores, organizes and provides access to information about (physical and virtual) sensors.</td>
</tr>
<tr>
<td><strong>(OpenIoT)</strong> Sensor Middleware</td>
<td>A part of the OpenIoT middleware platform that facilitates access to, collection and filtering of OpenIoT data streams.</td>
</tr>
<tr>
<td><strong>Global Sensor Networks (GSN)</strong></td>
<td>An open source sensor middleware platform enabling the development and deployment of sensor services with almost zero-programming effort.</td>
</tr>
<tr>
<td><strong>Data Stream</strong></td>
<td>A stream of digital information stemming from a physical or virtual sensor.</td>
</tr>
<tr>
<td><strong>Linked Sensor Data</strong></td>
<td>A set of Semantic Web technologies for exposing, sharing, and connecting sensor data, information, and knowledge.</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

1.1 Scope

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

OpenIoT deliverable D4.3.1 corresponds to the first release of the open source implementation of the OpenIoT middleware framework (i.e., the core OpenIoT middleware platform), which includes a wide range of components of the OpenIoT architecture. Deliverable D4.3.1 includes both a prototype and the present document/report. The prototype implementation of the deliverable is available at the Github infrastructure which is devoted to the project and which is available at: https://github.com/OpenIoTOrg/openiot.

This Github infrastructure contains the prototype implementation of the various components that comprise this initial 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 first OpenIoT platform 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 baseline instructions for using/running the component based on source code available within the Github.

---

In addition, to detail 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 workpackages 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.

A second version of this deliverable will be released during the third year of the project’s lifetime, with a view to documenting the evolution of the OpenIoT open source project as part of the second (and final) release of the software. The final release of the open source software will provide a more mature and detailed implementation of the OpenIoT architecture.

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 also willing to contribute to the OpenIoT project. For these members, the deliverable can serve as a basis for understanding the technical implementation of the components that comprise the first 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.

1.3 Summary

In principle 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, along with a description of its instantiation in the scope of the first 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 first open source release.

Note that the report includes a concluding section, which also provides an outlook towards the second release of the deliverable.

1.4 Structure

The deliverable is structured as follows:

- Section 2 following this introductory section 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, and accordingly illustrates how the components of the first release map to the architecture.

- Section 3 is devoted to the presentation of various components within the first 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.

- Section 4 describes the source code of the open source release in terms of structure and availability.

- Section 5 presents the ever important third party components and libraries, which are needed in order to deploy and run various OpenIoT components. The third-party components include platforms, containers and tools.

- Section 6 is the concluding section of the document. In addition to drawing conclusions this last section provides an outlook to the second (and final) release of the core OpenIoT platform.
2 OVERVIEW OF OPEN SOURCE RELEASE ARCHITECTURE

2.1 Overview of OpenIoT Architecture

OpenIoT Architecture is comprised by seven main elements as depicted in Figure 1. The Sensor Middleware, the Cloud Data Storage, 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. The main core components have been introduced first in previous project documents (i.e D2.1\textsuperscript{2}, D2.2\textsuperscript{3}) and described with more detailed in terms of service architecture functional blocks in D4.1\textsuperscript{4}. In this section an overview of those components with accurate refinements in functionality is included.

- 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

---

\textsuperscript{2} D41 Service Delivery Environment Formulation Strategies
\textsuperscript{3} D22 OpenIoT Platform Requirements and Technical Specifications
\textsuperscript{4} D23 OpenIoT Detailed Architecture and Proof-of-Concept Specifications
Middleware is deployed on the basis of one or more distributed instances (nodes), which may belong to different administrative entities. The prototype implementation of the OpenIoT platform uses the GSN sensor middleware that has been extended and called X-GSN (Extended GSN).

• The **Cloud Data Storage** (Linked Stream Middleware Light, LSM-Light) enables the storage of data streams stemming from the sensor middleware thereby acting as a cloud database. The cloud infrastructure stores also the metadata required for the operation of the OpenIoT platforms (functional data). In addition to data streams and metadata, the cloud could also host computational (software) components of the OpenIoT platform (i.e. Schedules and SD&UM) in order to benefit from the elasticity, scalability and performance characteristics of the cloud. The prototype 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 **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 service provision.

• The **Service Delivery & Utility Manager** performs a dual role. On the one hand, it combines the data streams as indicated by service workflows within the OpenIoT system in order to deliver the requested service (with the help of the SPARQL query provided by the Scheduler). To this end, this component makes use of the service description and resources identified and reserved by the Scheduler component. On the other hand, this component acts as a service metering facility which keeps track of utility metrics for each individual service. This metering functionality will be accordingly used to drive functionalities such as accounting, billing and utility-driven resource optimization. Such functionalities are essential in the scope of a utility (pay-as-you-go) computing paradigm, such as the one promoted by OpenIoT.

• 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 Global Scheduler. This component may be accompanied by a GUI (Graphical User Interface).

• The **Request Presentation** component, which is in charge of the visualization of the outputs of a service that is provided within the OpenIoT platform. This component selects mashups from an appropriate library in order to facilitate service presentation. It is expected that service integrators implementing/integrating solutions with the OpenIoT platform are likely to enhance or even override the functionality of this component on the basis of a GUI pertaining to their solution.

• The **Configuration and Monitoring** component, which enables management and configuration of functionalities over the sensors and the (OpenIoT) services that are deployed within the OpenIoT platform. It is also supported by a GUI.
2.2 OpenIoT Architecture Proof of Concept

The OpenIoT project provides a proof-of-concept (PoC) implementation which presents a minimal set of components that demonstrates the basic workflows of OpenIoT architecture services. This implementation is used as the “skeleton” of the final platform and is available through the OpenIoT open source portal. The objectives of this implementation are the following:

- To provide the first integrated version of the OpenIoT software to the open source community.
- To involve the first users, from outside the consortium, to get a feedback from the user/developer regarding the OpenIoT architecture.
- To bootstrap the OpenIoT open source community.

Figure 2 below illustrates the high level view of the functional blocks of the main components of the OpenIoT Architecture, which were listed above.

![Figure 2. OpenIoT Architecture Overview (functional view)]
In addition, the first PoC implementation will enable the identification of issues, problems and potential needs for platform redesign that may arise during platform usage. The PoC implementation requires a minimal set of modules and services, from the ones described in high level above and in detail in deliverable D2.3 (OpenIoT Detailed Architecture and Proof-of-Concept Specifications), to be implemented. A high level view of the core modules and the main entities involved are depicted in Figure 3. The PoC architecture modules respectively provided are:

- **Directory Service**: The directory service is provided in the form of an RDF Store (annotated database using RDF format) also called triple store that is accessed through the LSM-Light module.

- **GSN-RDF/SSN integration (X-GSN)**: The original GSN implementation has been upgraded to an OpenIoT version, named Extended GSN (X-GSN), which can announce sensors and semantically annotate received data streams.

- **Scheduler**: The Scheduler provides a basic implementation of its complete API (i.e. sensor discovery and application/service management) which mainly interacts with the “serviceDescription” entity and the Request Definition UI.

- **Service Delivery & Utility Manager (SD&UM)**: The SD&UM provides the basic implementation of its complete API which mainly interacts with the “serviceDescription” entity, “virtualSensorsDataStorage” entity and the Request Presentation UI.

- **Request Definition UI**: A basic functionality of discovering sensor models and building service requests is supported. The module which the Request Definition UI interacts with is the Global Scheduler.

- **Request Presentation UI**: A basic functionality of requesting of available services of a specific user, polling for data regarding a specific service and visualizing them is supported. The module which the Request presentation UI interacts with is the SD&UM.

---

Figure 3. OpenIoT PoC Modules and Entities
2.2.1 Data Flow

Figure 4 below depicts the OpenIoT PoC implementation by also providing a complete example of the platform’s data and service flow from the deployment, configuration and presentation. The flow can be outlined as follows:

- X-GSN nodes are “announcing” 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 (Step 0).
- A User requests from the Scheduler (Step 1) 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 (Step 2) a combination of queries (SPARQL) to fulfil the previously user specified query provided by the previous step.
- The Directory Service retrieves the data (the “availableSensors” entity) and replies back to the Scheduler (Step 3) with the available sensor models.
- The reply is forwarded to the Request Definition UI from the Scheduler (Step 4) and the retrieved information is provided to the User.
- 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 Object) object (Figure 21, Figure 22). This OSDSpec is then pushed to the Scheduler with the help of “registerService” (Step 5).
- The Scheduler analyses the received OSDSpec and sends the request (Step 6) to the Directory Service (“serviceDescription” entity).
- 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 (Steps 7, 8, 9 and 10) all the registered applications/services related to a specific User (available at the “serviceDescription” entity).
- Having selected a specific 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 (Step 11).
- The SD&UM requests (SD&UM’s “getService”) from the Directory Service to retrieve (Step 12) all related information for the specific Service (available at the “serviceDescription” entity).
• The Directory Service provides this information to the SD&UM (Step 13).
• The SD&UM analyses the retrieved information, which are available in the replied OSMO object (Figure 22), and forwards the included SPARQL script (Which has been created by the Request Definition UI (step 5) and stored by the Scheduler (Step 6)) to the Directory Service SPARQL interface (Step 14).
• The result is sent from the Directory Service to the SD&UM (Step 15), in a SparqResultsDoc\textsuperscript{5} format. Then the SD&UM forwards it to the Request Presentation (Step 16) within a SdumServiceResultSet (Figure 23 below) object that also includes information on how these data should be presented.

\textsuperscript{5} http://www.w3.org/TR/rdf-sparql-XMLres/#defn-srd
3 OPENIOT COMPONENTS DEVELOP/DEPLOY/USE

This section is devoted to the presentation of the components comprising the first 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 Wiki space under the Documentation section.

3.1 Scheduler

As already stated, 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>
<thead>
<tr>
<th>Table 1. List of primitives comprising the implemented Scheduler API</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;interface&gt;&gt;</td>
</tr>
<tr>
<td>SchedulerInterface</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>discoverSensors</td>
</tr>
<tr>
<td>(userID:String, longitude:double, latitude:double, radius:float): SensorTypes</td>
</tr>
<tr>
<td>registerApp</td>
</tr>
<tr>
<td>(osdSpec: OSDSpec): String</td>
</tr>
<tr>
<td>unregisterApp</td>
</tr>
<tr>
<td>(String applicationID): void</td>
</tr>
<tr>
<td>updateApp</td>
</tr>
<tr>
<td>(osdSpec: OSDSpec): void</td>
</tr>
<tr>
<td>getApplication</td>
</tr>
<tr>
<td>(applicationID: String): OAMO</td>
</tr>
<tr>
<td>getService</td>
</tr>
<tr>
<td>(serviceID: String): OSMO</td>
</tr>
<tr>
<td>getAvailableAppIDs</td>
</tr>
<tr>
<td>(userID: String): DescriptiveIDs</td>
</tr>
<tr>
<td>getAvailableServiceIDs</td>
</tr>
<tr>
<td>(applicationID: String): DescriptiveIDs</td>
</tr>
<tr>
<td>getAvailableApps</td>
</tr>
<tr>
<td>(userID: String): OSDSpec</td>
</tr>
<tr>
<td>getUser</td>
</tr>
<tr>
<td>(userID: String): OpenIoTUser</td>
</tr>
</tbody>
</table>

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

6 https://github.com/OpenIotOrg/openiot/wiki

7 https://github.com/OpenIotOrg/openiot/wiki/Documentation
Table 2. Implemented Scheduler API definition

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Input</th>
<th>Output</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>discoverSensors</td>
<td>String userID, double longitude, double latitude, float radius</td>
<td>SensorTypes</td>
<td>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.</td>
</tr>
<tr>
<td>registerApp</td>
<td>OSDSpec osdSpec</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>unregisterApp</td>
<td>String applicationID</td>
<td>void</td>
<td>Used to unregister/delete a registered/running service. Requires as input the Application ID.</td>
</tr>
<tr>
<td>updateApp</td>
<td>osdSpec: OSDSpec</td>
<td>void</td>
<td>Used to update a registered service. Requires as input an OSD Specification.</td>
</tr>
<tr>
<td>getService</td>
<td>String serviceID</td>
<td>OSMO</td>
<td>Used to retrieve the description (OSMO) of an available service. Requires as input a Service ID.</td>
</tr>
<tr>
<td>getApplication</td>
<td>String applicationID</td>
<td>OAMO</td>
<td>Used to retrieve the description (OAMO) of an available Application. Requires as input the Application ID</td>
</tr>
</tbody>
</table>
### 3.1.2 Download, Deploy & 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 and LSM-Light client library 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.

---

<table>
<thead>
<tr>
<th>Method</th>
<th>Input Parameters</th>
<th>Returns/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getAvailableAppIDs</code></td>
<td>String <code>userID</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>getAvailableServiceIDs</code></td>
<td>String <code>serviceID</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>getAvailableApps</code></td>
<td>String <code>userID</code></td>
<td>OSDSpec Used to retrieve the services defined by a user. It returns an OpenIoT Service Description Specification. Requires as input a User ID.</td>
</tr>
<tr>
<td><code>getUser</code></td>
<td>String <code>userID</code></td>
<td>OpenIoUser Used to retrieve user information for implementing access control mechanisms.</td>
</tr>
</tbody>
</table>

The Schemata as long as the schema diagrams of OSDSpec, OAMO, OSMO, SensorTypes and DescriptiveID can be found in “APPENDIX I – SCHEMATA” section below.
To install LSM-Light client library go to OpenIoT GitHub under
utils/lib/lsmApiLibraryMavenInstall\(^8\) and download “lsmlibs.jar”. Go to the folder you
downloaded the jar file thru the command line and execute the following maven command:

```
mvn install:install-file -Dfile=lsmlibs.jar -DgroupId=org.openiot -DartifactId=lsm.api -Dversion=0.0.1 -Dpackaging=jar
```

### 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/OpenIotOrg/openiot.git](https://github.com/OpenIotOrg/openiot.git)
- SSH: `git@github.com:OpenIotOrg/openiot.git`

The scheduler is available under the “openiot /modules /scheduler/” folder.

### 3.1.2.1.3 Deploy from the source code

If you have not yet done so, you must Configure Maven before testing the scheduler
deployment. After that:
- 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 the Scheduler
  - 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.
  1. Make sure you have started the JBoss Server as described above.
  2. Open a command line and navigate to the root directory of the scheduler
     Project.
  3. Type this command to build and deploy the archive:
     1. `mvn clean package jboss-as:deploy`
  4. This will deploy target/scheduler.core.war to the running instance of the
     server.

\(^8\) [https://github.com/OpenIotOrg/openiot/tree/develop/utils/lib/lsmApiLibraryMavenInstall](https://github.com/OpenIotOrg/openiot/tree/develop/utils/lib/lsmApiLibraryMavenInstall)
• Access the application
  o The application will be running at the following URL:
    http://localhost:8080/scheduler.core/.

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

3.1.2.1.4 Run in Eclipse

3.1.2.1.4.1 Integrating and Starting JBoss server

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

3.1.2.1.4.2 Integrating and deploying Scheduler

To integrate and deploy the Scheduler 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 scheduler’s source code directory that has been previously downloaded.
  3. Choose the scheduler.core and click the Finish button.
  4. Right click on the “scheduler.core” project and choose “Run As>Maven Build…”
  5. Insert the following to:
     a. Goals: “clean package jboss-as:deploy”
     b. Profiles: “arq-jbossas-remote”
     c. Name: “scheduler.core package-deploy” (or your preferred name)
  6. Click the Run button (the JBoss Server should be already running). The project will automatically build itself, get deployed and run at the JBoss AS running instance. From now on this configuration should be available at the Eclipse Run Configurations under Maven Build.
To undeploy the scheduler from the running instance of the JBoss AS follow the steps below:

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

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

3.1.2.2 User

3.1.2.2.1 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. You can download the binaries through the OpenIoT Wiki ⁹ under the Users>Downloads ¹⁰ section.

3.1.2.2.2 Deployment/Undeployment

3.1.2.2.2.1 JBoss AS 6.0

**Deploy:** To deploy the scheduler on JBoss AS 6.0, copy the “scheduler.core.war” to the server's “deploy” directory.

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

---

⁹ [https://github.com/OpenIotOrg/openiot/wiki](https://github.com/OpenIotOrg/openiot/wiki)
3.1.2.2.2 JBoss AS 7.0

**Deploy:** To deploy the scheduler on JBoss AS 7.0, copy the “scheduler.core.war” to the server’s “standalone/deployments” directory.

**Undeploy:** To undeploy the application, you need to remove the “.deployed” marker file that is generated upon successful deployment of the scheduler 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](https://docs.jboss.org/author/display/AS7/Application+deployment)

3.1.2.2.3 Manual

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:
- 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

As already stated, 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

<table>
<thead>
<tr>
<th></th>
<th>Input</th>
<th>Output</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>pollForReport</td>
<td>String serviceID</td>
<td>SdumServiceResultSet</td>
<td>Invokes a previously defined Service having the specified applicationID. This call will produce only one Result Set.</td>
</tr>
<tr>
<td>getService</td>
<td>String serviceID</td>
<td>OSMO</td>
<td>Used to retrieve the description (OSMO) of an available service. Requires as input a Service ID.</td>
</tr>
<tr>
<td>getApplication</td>
<td>String applicationID</td>
<td>OAMO</td>
<td>Used to retrieve the description (OAMO) of an available Application. Requires as input an Application ID.</td>
</tr>
</tbody>
</table>
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 long as the schema diagrams of SdumServiceResultSet, OAMO, OSMO and DescriptiveID can be found in “APPENDIX I – SCHEMATA” section below.

3.2.2 Download, Deploy & Run

3.2.2.1 Developer

3.2.2.1.1 System requirements

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

To install LSM-Light client library go to OpenIoT GitHub under utils/lib/lsmApiLibraryMavenInstall11 and download “lsmlibs.jar”. Go to the folder you downloaded the jar file thru the command line and execute the following maven command:

```
mvn install:install-file -Dfile=lsmlibs.jar -DgroupId=org.openiot -DartifactId=lsm.api -Dversion=0.0.1 -Dpackaging=jar
```

11 https://github.com/OpenIoTOrg/openiot/tree/develop/utils/lib/lsmApiLibraryMavenInstall
3.2.2.1.2 Download

To download Service Delivery & Utility 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 SD&UM is available under the "openiot /modules /sdum/" folder.

3.2.2.1.3 Deploy from the source code

If you have not yet done so, you must Configure Maven before testing the scheduler deployment. After that do the following:

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

- Build and Deploy the SD&UM
  o NOTE: The following build command assumes you have configured your Maven user settings. If you have not done so, you must include Maven setting arguments on the command line.
  1. Make sure you have started the JBoss Server as described above.
  2. Open a command line and navigate to the root directory of the sdum Project.
  3. Type this command to build and deploy the archive:
     o mvn clean package jboss-as:deploy
  4. This will deploy target/sdum.core.war to the running instance of the server.

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

- Undeploy the archive
  1. Make sure you have started the JBoss Server as described above.
  2. Open a command line and navigate to the root directory of the sdum Project.
  3. When you are finished testing, type this command to undeploy the archive:
     o mvn jboss-as:undeploy
3.2.2.1.4 Run in Eclipse

3.2.2.1.4.1 Integrating and Starting JBoss server

You can start JBoss Application Server and deploy the Scheduler 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

3.2.2.1.4.2 Integrating and deploying SD&UM

To integrate and deploy the SD&UM 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 SD&UM’s source code directory that has been previously downloaded.
3. Choose the sdum.core and click the Finish button.
4. Right click on the “sdum.core” project and choose “Run As>Maven Build…”
5. Insert into:
   a. Goals: “clean package jboss-as:deploy”
   b. Profiles: “arq-jbossas-remote”
   c. Name: “sdum.core 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 SD&UM from the running instance of the JBoss AS, follow the steps below:

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

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

3.2.2.2.1 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. You can download the binaries through the OpenIoT Wiki under the Users>Downloads section.

3.2.2.2.2 Deployment/Undeployment

3.2.2.2.2.1 JBoss AS 6.0

**Deploy**: To deploy the SD&UM on the JBos AS6.0, copy the “sdum.core.war” to the server’s “deploy” directory.

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

3.2.2.2.2.2 JBoss AS 7.0

**Deploy**: To deploy the SD&UM on JBos AS 7.0, copy the “scheduler.core.war” to the server’s “standalone/deployments” directory.

**Undeploy**: To undeploy the application, you need to remove the “.deployed” marker file that is generated upon successful deployment of the SD&UM 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](https://docs.jboss.org/author/display/AS7/Application+deployment)

---

3.2.2.2.3 Manual

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 “3.5.2 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:
  o http://localhost:8080/sdum.core/rest/services/

- Poll for Report:
  o http://localhost:8080/sdum.core/rest/services/pollforreport

- Get Service Status:
  o http://localhost:8080/sdum.core/rest/services/getServiceStatus

- Get Application:
  o http://localhost:8080/sdum.core/rest/services/getApplication

- Get Service:
  o http://localhost:8080/sdum.core/rest/services/getService

- Get User:
  o http://localhost:8080/sdum.core/rest/services/getUser

- Get Available Application IDs:
  o http://localhost:8080/sdum.core/rest/services/getAvailableAppIDs

- Get Available Service IDs:
  o http://localhost:8080/sdum.core/rest/services/getAvailableServiceIDs
3.3 **Linked Stream Middleware Light (LSM-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 prototype 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>
<thead>
<tr>
<th>Service Name</th>
<th>Input</th>
<th>Output</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSensorById</td>
<td>String sensorID</td>
<td>Sensor</td>
<td>Used to retrieve an existing sensor from LSM by sending a request. Requires</td>
</tr>
<tr>
<td>Method</td>
<td>Input Parameters</td>
<td>Output Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getSensorBySource</td>
<td>String sensorSource</td>
<td>Sensor</td>
<td>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.</td>
</tr>
<tr>
<td>sensorAdd</td>
<td>Sensor sensor</td>
<td>boolean</td>
<td>Used to register a new sensor into LSM. Requires as input a Sensor class instance. This method returns a boolean indicating whether the sensor was successfully added or not (true or false respectively).</td>
</tr>
<tr>
<td>sensorAdd</td>
<td>String triples</td>
<td>boolean</td>
<td>Used to register a new sensor into LSM. Requires as input triples (if the sensor data is already in N3 format). This method returns a boolean indicating whether the sensor was successfully added or not (true or false respectively).</td>
</tr>
<tr>
<td>sensorDataUpdate</td>
<td>Observation observation</td>
<td>boolean</td>
<td>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 boolean indicating whether the observed data was successfully updated or not (true or false respectively).</td>
</tr>
<tr>
<td>sensorDataUpdate</td>
<td>String triples</td>
<td>boolean</td>
<td>Used to update the latest observed data generated by a sensor. Requires as input triples that were 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.</td>
</tr>
<tr>
<td>Method</td>
<td>Parameters</td>
<td>Returns</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------</td>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getUser</td>
<td>String userID</td>
<td>LSMUser</td>
<td>Used to retrieve the user’s information for implementing access control mechanisms.</td>
</tr>
<tr>
<td>setUser</td>
<td>User user</td>
<td>void</td>
<td>Used to register a new user into LSM. Requires as input a User object that includes a username and password.</td>
</tr>
<tr>
<td>deleteTriples</td>
<td>String graphURL</td>
<td>boolean</td>
<td>Used to clear all the triple data of a specific graph. Requires as input the graphURL. This method returns a boolean indicating whether the data were successfully removed or not (true or false respectively). Note that the data cannot be restored after this method is called.</td>
</tr>
<tr>
<td>deleteTriples</td>
<td>String graphURL, String triples</td>
<td>boolean</td>
<td>Used to clear specific triples from a specific graph. Requires as input the graphURL and triple patterns. This method returns a boolean indicating whether the data were successfully removed or not (true or false respectively). Note that the data cannot be restored after this method is called.</td>
</tr>
<tr>
<td>transformTriples</td>
<td></td>
<td></td>
<td>transformed from raw data to N3 format and includes all the available observed data. This method returns a boolean indicating whether the observed data was successfully updated or not (true or false respectively).</td>
</tr>
</tbody>
</table>
3.3.2 Download, Deploy 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 several required components 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 Apache Tomcat server

For the server side of LSM-LIGHT, we use Java Servlets, which are implemented on a Tomcat server. Please visit “http://tomcat.apache.org/” to download and install the latest Tomcat server version.

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/OpenIotOrg/openiot.git

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

3.3.2.1.3 Deploy from source code

LSM-LIGHT relies on Maven or its deployment. If you have not yet done so, you must Configure Maven before testing deployment.

LSM-LIGHT is deployed on Tomcat web server, thus, once Maven is successfully installed, you can deploy maven based LSM-LIGHT to Tomcat by using Maven-Tomcat plugin following the directions provided below. A non-maven solution is provided in User-Deployment section.
The steps to deploy LSM-LIGHT server side are the following:

1) **Tomcat Authentication:**
   First, add a user with administrator access rights for Tomcat. For doing so, please edit this file — “%TOMCAT_PATH%/conf/tomcat-users.xml”.
   
   File: tomcat-users.xml
   
   ```xml
   <?xml version='1.0' encoding='utf-8'?>
   <tomcat-users>
     <role rolename="manager"/>
     <role rolename="admin"/>
     <user username="admin" password="password" roles="admin,manager"/>
   </tomcat-users>
   ```
   
   You will use the user “admin” to do the deployment.

2) **Maven Authentication:**
   On Maven side, you need to add the same user authentication information in “%MAVEN_PATH%/conf/settings.xml”.
   
   File: settings.xml
   
   ```xml
   //...
   <server>
     <id>TomcatServer</id>
     <username>admin</username>
     <password>password</password>
   </server>
   //...
   ```

3) **Maven-Tomcat-Plugin:**
   Declare “Maven-Tomcat plugin” and related Tomcat server detail in your pom.xml file.
   
   ```xml
   <plugin>
     <groupId>org.codehaus.mojo</groupId>
     <artifactId>tomcat-maven-plugin</artifactId>
     <configuration>
       <url>http://127.0.0.1:8080/manager/</url>
       <server>TomcatServer</server>
       <path>/lsm-light.server</path>
     </configuration>
   </plugin>
   ```

4) **Deploy to Tomcat:**
   Issue “mvn tomcat:deploy” to package your project in a WAR file, and deploy it to the Tomcat server. To verify it, just access to the Tomcat’s manager page and make sure “/lsm-light.server” path exists.
   
   URL: [http://127.0.0.1:8080/manager/] or [http://localhost:8080/manager/]
3.3.2.1.4 Triple storage server connection property file

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

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection.driver_class</td>
<td>Java JDBC class name</td>
<td>virtuoso.jdbc4.Driver</td>
</tr>
<tr>
<td>connection.url</td>
<td>Server host</td>
<td>jdbc:virtuoso://localhost:1111/</td>
</tr>
<tr>
<td>connection.username</td>
<td>Username connection</td>
<td>dba</td>
</tr>
<tr>
<td>connection.password</td>
<td>Password connection</td>
<td>dba</td>
</tr>
<tr>
<td>minConnection</td>
<td>Minimum number of connection</td>
<td>5</td>
</tr>
<tr>
<td>maxConnection</td>
<td>Maximum number of connection</td>
<td>10</td>
</tr>
<tr>
<td>acquireRetryAttempts</td>
<td>After attempting to acquire a connection and failing, try to connect these many times before giving up.</td>
<td>5</td>
</tr>
</tbody>
</table>

3.3.2.1.5 Run in Eclipse

You can also start the server and deploy the LSM-LIGHT from Eclipse that has Maven and Tomcat Server plugins installed.

1) Install m4eclipse\(^\text{14}\) in Eclipse

2) Install Tomcat server in Eclipse: Please follow the steps below to install Tomcat Server 7 into Eclipse (Juno):

\(^{14}\) [http://m4eclipse.sourceforge.net/installation.html](http://m4eclipse.sourceforge.net/installation.html)
[3] Choose the Server tab from the bottom panel of the Eclipse Juno Window.
[4] Click the New Server Wizard... (Figure 5)

![Figure 5. Eclipse Juno - New Server Wizard](image)

[5] Ensure that a Tomcat Server, e.g., XAMPP Tomcat is running\(^\text{15}\). Choose your server and click Next. (Figure 6)

![Figure 6. Eclipse Juno – Choosing/Registering your Server](image)

[6] Set your Tomcat Server path. You can use ".\.\" to set a relative path starting from Eclipse Root. Click Finish. (Figure 7)

\(^{15}\) [http://setup-steps.blogspot.gr/2013/01/xampp-for-windows.html](http://setup-steps.blogspot.gr/2013/01/xampp-for-windows.html)
Figure 7. Eclipse Juno – Setting your Server Path

[7] Tomcat is now registered and available in the Servers tab. Select the server and click Start (or Right-click and select Start) to start the server.

Figure 8. Eclipse Juno – Selecting your Now Registered Server

[8] If the server has been started and is running well, the Server Stop Button should be active and appears in red colour. (Figure 9)

Figure 9. Eclipse Juno – Registered Server Running

3.3.2.1.6 LSM-Light Sensors Data Manipulation

These examples below illustrate some functionalities of LSM-Light api (lsm-light.client module). By following this, the users will know how to communicate with 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

```java
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.setSource("http://opensensedata.epfl.ch:22002/gsn?REQUEST=113&name=lausanne_1057");
            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
             */
            User user = new User();
        }
    }
}
```
user.setUsername("sofiane");
user.setPass("sofiane");
sensor.setUser(user);

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

//set user information for authentication
lsmStore.setUser(user);

//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.deri.ie/resource/8a82919d3264f4ac013264f4e14501c0" is the sensorURL of lausanne_1057 sensor
obs.setSensor("http://lsm.deri.ie/resource/8a82919d3264f4ac013264f4e14501c0");

//set time when the observation was observed. In this example, the time is current local time.
obstime = 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");
obvTem.addReading(obvTem);

ObservedProperty obvCO = new ObservedProperty();
obvCO.setObservationId(obs.getId());
obvCO.setPropertyType(ObsConstant.HUMIDITY);
obvCO.setValue(0.0366300366300366);
obvCO.setUnit("C");
obs.addReading(obvCO);
ob.setDatagraph("http://lsm.deri.ie/OpenIoT/new/sensordata#");
ob.setMetaGraph("http://lsm.deri.ie/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**

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

**Example 4 - Delete sensor data**

```java
/**
 * remove sensor
 * @param sensorURL
 */
lsmStore.sensorDelete("http://lsm.deri.ie/resource/8a82919d3264f4ac013264f4e14501c0","http://lsm.deri.ie/OpenIoT/demo/sensormeta#";

/**
 * delete all reading data of specific sensor
 * @param sensorURL
 */
lsmStore.deleteAllReadings("http://lsm.deri.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.deri.ie/resource/8a82919d3264f4ac013264f4e14501c0", "<=", fromDate, null);
```
3.3.2.2 User

3.3.2.2.1 System requirements

LSM-LIGHT.client requires Java JDK 1.6 or later version. However, we recommend using Java JDK 1.7.

3.3.2.2.2 Libraries requirements

- Dom4j (download here: http://sourceforge.net/projects/dom4j/)

3.3.2.2.3 Download

LSM-LIGHT.client can be downloaded as Java JAR from the LSM Home page or OpenIoT binary distribution repository: https://github.com/OpenIoTOrg/openiot/tree/develop/utils/lib

3.3.2.2.4 Deployment

3.3.2.2.4.1 LSM-LIGHT server side deployment

When LSM-LIGHT maven project is correctly deployed (exported to a war file), if you already have your Tomcat server running and the server cannot be stopped, please follow this procedure to deploy LSM-LIGHT without any interruption to your server:

1. Change to the root directory of your web application. In this case the root directory would be TOMCAT_HOME/webapps/onjava/.
2. Copy the lsm-light.server WAR file, lsm-light.server.war, to the TOMCAT_HOME/webapps directory.

If you're deploying this WAR file to the Tomcat installation that you were developing in, then you will need to back up your development directory and remove it from the TOMCAT_HOME/webapps directory.

3. Go to Tomcat Manager web page and refresh it.

3.3.2.2.4.2 LSM-LIGHT client side (LSM API) deployment

The LSM API is used by the Scheduler, SDUM and X-GSN. In order to communicate with your LSM server side, please attach the LSM API into your project's library and follow the examples mentioned in Developer-Manual section (3.3.2.1) to insert and retrieve OpenIoT data from LSM server.
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 10).

![Diagram of data acquisition network in X-GSN](image_url)

Figure 10. Data acquisition network in X-GSN

A set of wrappers allow 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 to a SQL data structure with fields that have a name and type. X-GSN is then able to filter this using its 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) that are 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 website for displaying the average measured wind 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 Download, Deploy and Run

3.4.2.1 Developer

3.4.2.1.1 System requirements
X-GSN requires Java JDK 1.6 to run and is not compatible with higher Java versions. If you are using Java 1.7 or higher, you have to explicitly compile X-GSN with 1.6 as the source and target versions. If you use the Maven build file, this will be done 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/OpenIoT_org/openiot.git
X-GSN module can be found under modules/x-gsn.

3.4.2.1.3 Deploy from source code
X-GSN relies on Maven or its deployment. If you have not yet done so, you must configure Maven before testing deployment.

Once Maven is correctly deployed, you can generate the X-GSN jar file by calling 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 to run from command line.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gsn-start</td>
<td>Starts the X-GSN server</td>
</tr>
<tr>
<td>gsn-stop</td>
<td>Stops the X-GSN server</td>
</tr>
<tr>
<td>lsm-register</td>
<td>Registers sensor metadata to LSM</td>
</tr>
</tbody>
</table>

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](http://localhost:22001)

3.4.2.2 User

3.4.2.2.1 System requirements

X-GSN requires Java JDK 1.6 to run. If you have a higher version of Java, there is no problem as the binary version of X-GSN is already compiled with the right version.

3.4.2.2.2 Download

X-GSN (xgsn.jar) can be downloaded as a Java JAR from the OpenIoT binary distribution repository[^16].

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 Manual

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.

[^16]: [https://github.com/OpenIoTOrg/openiot/wiki/Downloads](https://github.com/OpenIoTOrg/openiot/wiki/Downloads)
3.4.2.4.2 Semantic annotation of sensor data

3.4.2.4.2.1 Configuring GSN for integration with LSM

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

```plaintext
username=user
password=pass
metaGraph=http://lsm.deri.ie/OpenIoT/demo/sensormeta#
dataGraph=http://lsm.deri.ie/OpenIoT/demo/sensordata#
```

3.4.2.4.2.2 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.xml.metadata`. Sample virtual sensors with associated metadata can be found in (virtual-sensors/samples/LSM).

Table 8 below shows a sample metadata file.

<table>
<thead>
<tr>
<th>Sensor Name</th>
<th>Data Graph</th>
<th>Meta Graph</th>
<th>Source</th>
<th>Author</th>
<th>Sensor Type</th>
<th>Source Type</th>
<th>Information</th>
<th>Sensor ID</th>
<th>Registered</th>
<th>Field Humidity</th>
<th>Field Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.4.2.4.2.3 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.xml.metadata
lsm-register.bat virtual-sensors\opensense1.xml.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.deri.ie/resource/557635232734257
```

3.4.2.4.2.4 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.

3.5 User Interfaces

3.5.1 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). These applications are able to group a collection of different services (OSMOs) which comprises/describes a real life application (i.e. weather reports). This enables end-users to manage (describe/register/edit/update) different (unrelated) applications from a single point.
All modelled services are stored by the OpenIoT Scheduler and are automatically loaded when a user accesses the web application.

### 3.5.1.1 Main Released Functionalities & Services

Figure 11 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 on 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 11. Request Definition User Interface (UI)](image-url)
### 3.5.1.2 Reset or refresh applications

To clear all applications and services click on the “File” menu and select the “Reset applications” option. To reload all applications and services from the server, click on the “File” menu and select the “Reload applications” option.

### 3.5.1.3 Creating a new application

To create a new application, select the “New application” option from the “File” menu. The new service dialog will appear. Enter a name for the application and optionally add a description for it. Choose apply to create a blank application and open it for editing.

### 3.5.1.4 Editing an existing application

To edit an existing application click on the “File” menu, then select the application name from the “Open application” sub-menu. The system will load the application and open it for editing.

### 3.5.1.5 Modelling the service graph of an application

A service graph describing a particular application can be created by dragging nodes from the node-toolbox into the application workspace, setting them up and wiring them together.

Each graph node is colour-coded to indicate its function and provides input and output endpoints to facilitate connections. Each endpoint belongs to one or more scopes while its position on the rendered node is dictated by its function (inputs on the left, outputs on the right). The system will only allow connections between endpoints that have common scopes. Connections between nodes may be established by clicking on an output endpoint and dragging a connection to another node’s input endpoint. While a connection is dragged, the system will automatically highlight the endpoints that can serve as the connection’s destination.

A service graph visualizes the information flow from a source (typically a sensor type) to a sink (a visualization widget). Therefore, all service graphs should contain at least one source node and one sink node. Other node types can be injected between the source and the sink to manipulate the data (i.e. perform an aggregation or filtering function). The available node types are described in the following sections.

#### 3.5.1.5.1 Data source node

Data source nodes model the sensor types available for querying via the OpenIoT middle-ware. In order to populate the list of available sensors, a sensor discovery query has to be performed. By clicking the search button in the node toolbox, a sensor discovery dialog will appear (Figure 12).
To search for sensors within a specific area, fill in the location textbox and click on the lookup button. The map view will be updated with the selected location. Click on the map to select the centre of the search query and fill in the search radius textbox. Then click on the “Find sensors” button. The node toolbox will be updated with the list of available sensor types (classes) that match the search criteria.

![Image of Sensor Discovery Dialog](image.png)

Figure 12. The Sensor Discovery Dialog

Each sensor node represents all the sensor instances of a particular sensor type that are available at a specific location. When dragged into the workspace, the node encodes the search criteria that were used for locating it into its embedded properties. By dragging the appropriate sensor nodes into the workspace we can model graphs that process similar data from different locations.

A sensor node example and its properties are shown on Figure 13. All available sensor attributes appear as output endpoints. The node provides an additional input endpoint for connecting a selection filter node. The functions of this special node are described in the following section.
3.5.1.5.2 Selection filter node

Sometimes, it is desirable to process data within a specific time window. This node allows us to limit the data records that will be processed using time-based criteria. Once connected to a sensor node, the selection filter node will expose its recordTime endpoint. This endpoint can be connected to a comparator node that describes filter parameters.

3.5.1.5.3 Comparator nodes

These nodes can be connected to a selection filter node to define a time-based filter. Each node's filter parameters are exposed as node properties. The following nodes are provided:

- **Between(date)**. Ensures that the processed records fall between two specific dates.
- **Compare(abs. date)**. Ensures that the processed records satisfy the condition “recordTime operator userDate”, where operator is a user-selected operator (less, less or equal, equal, greater or equal, greater) and userDate a user-selected date.
- **Compare (rel. date)**. Ensures that the processed records satisfy the condition “(NOW - recordTime) operator value timeUnit”, where operator is a user-selected operator, value is a numeric value used for the comparison and timeUnit is a user-selected time unit (seconds, minutes, hours, months, years). Figure 14 shows an example where the node is used to select data recorded within the last 5 hours.
3.5.1.5.4 Group node

This node allows users to partition the sensor data into time buckets using each observation's record time. To setup a group node, connect each attribute that should be grouped to the group node's input endpoint. Once a connection is made, a new output endpoint will appear on the right side of the group node. The new endpoint represents partitioned time buckets and may be connected to an aggregation node or directly to a compatible sink node.

To select the components of the observation's record time that will be used for grouping, select the group node and then click on the “Grouping options” button in the node's property pane. The grouping options dialog will appear (Figure 15). Select the components that will be used for grouping and click on the “Apply” button. An additional endpoint for each selected group field will appear at the right side of the group node (Figure 16). For each attribute, this node will generate an output tuple of the form: (grp_recordTime1, ..., grp_recordTimeN, valueSet)
Figure 15. Grouping Options Dialog

Figure 16. Grouping an attribute by record time
3.5.1.5.5 Aggregation nodes

Aggregation nodes apply an aggregation function to their inputs. The following aggregation functions are supported: min, max, count, average, sum. These nodes accept as input any output from a sensor, group or aggregation node.

3.5.1.5.6 Sink nodes

Sink nodes serve as the termination endpoint of a service graph. All service graphs should have at least one sink node connected in order to pass the service graph validation check. While a standard set of sink nodes is provided, additional sink nodes can be defined depending on the application. This process is described in detail in the following developer sections.

3.5.1.5.7 Line chart sink node

The line chart sink node will render a line chart widget that supports up to 5 data series. To setup the line chart, first select the number of series. Depending on the number of series selected, additional inputs \( y_i \) will appear on the node for connecting each series' Y axis data. Then select the type of data that will be plotted on the X axis. We support three types of data for the X axis:

- **Number.** The X axis will plot a numeric field. In this case, additional inputs \( x_i \) will appear on the node for connecting each series' X axis data. All X axis inputs only accept a single connection.
- **Date (result set).** In this mode, the X axis value will be automatically set to the system's timestamp when the query results arrive. In this case, no inputs for the X axis are available.
- **Date (observation).** This mode is designed to work together with a group node. In this case, additional inputs \( x_i \) will appear on the node for connecting each series' X axis timestamp data. These inputs accept multiple connections from a group node's grp_recordTime fields. Depending on which time fields where connected, the system will automatically generate a timestamp for the X axis while it processes incoming data. Figure 17 shows an example scenario where a line chart plots the max temperature for every day.
Figure 17. A service that plots the max temperature per day

3.5.1.5.8 Pie chart sink node

This node will render a pie chart and supports up to 10 series.
To setup the pie chart, first select the number of series. Depending on the number of series selected, additional inputs \( y_i \) will appear on the node for connecting each series' value.

3.5.1.5.9 Meter gauge sink node

This node will render a gauge with a dial indicating its current value.
To setup the gauge, enter its measure unit label and then select its minimum and maximum values. The node exposes a single input endpoint for connecting the gauge's value.

3.5.1.5.10 Map sink node

The map sink node renders a map containing markers indicating the location of sensors and optionally renders circle overlays to indicate the magnitude of a specific attribute from each sensor.
To set up this node you need to setup the map widget parameters. To do this, fill in the latitude, longitude and zoom level properties. Then select the type of overlays that should be rendered. We support the following overlay modes:
• Markers only. Renders a marker at each sensor's location.
• Circles only. Renders a circle at each sensor's location with a radius equal to the connected property's value scaled by the "max value" property. The max value scaling factor defaults to 1 and can be used to tweak the size of the generated circles.
• Markers and circles. This mode combines the previous modes.

3.5.1.5.11 Passthrough sink node

The passthrough sink node is a special node designed to be used by applications that need to bypass the request presentation layer and perform their own custom processing on the service data.

This node provides a simple sink with N inputs. The number of inputs is selected by editing the attribute count property of the node. Once the property is modified, the node will show the appropriate number of input endpoints( attr1, ..., attrN) depending on the property value.

Once the service has been registered, the application needs to manually invoke the SDUM to get the service results. When used via the request presentation layer, the bound attributes are rendered in tabular form.

3.5.1.6 Using variable property values

Some application scenarios involve queries where some of the parameters are not known a priori during service design. For example, a service that reports the availability of a specific resource (like a room) near a mobile user depends on the user's location which is only known when the query is to be executed.

To support such scenarios, we provide a mechanism for converting node properties into variables. This mechanism is supported for all nodes except the sink nodes.

To convert a property into a variable, click on the "Create variable" button next to the property. The variable editor dialog will appear (Figure 18). Select the "Convert to variable" check-box and then enter a name for the variable and specify its default value. Then click on the "Apply" button. As you can see in Figure 19, the property value has been replaced by the variable name and the field has become read-only.

This process can be undone by opening the variable editor and deselecting the convert to variable check-box.

Applications referencing services which contain variables should supply their values when they invoke the SD&UM. If a variable value is not specified, it is automatically set to the default value.
3.5.1.7 Workspace validation

Before an application can be saved, it has to pass the following workspace validation checks:

- Check for unconnected nodes.
- Check that all required node properties have been filled in.
- Check that all required node endpoints have been connected.
- Check that the service graph contains no closed loops.

In addition to the above checks, some node-specific checks are also performed. The output of a workspace validation is a list of problems (which should be fixed) and a set of warnings. The validation result appears in the console pane. By clicking on a problem or warning, the system will highlight its location on the service graph.

Validation runs automatically before saving an open application but it can also be manually invoked by clicking on the current application menu and selecting the “Validate design” button.

3.5.1.8 Deploy from source

Start 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:
  o For Linux: JBOSS_HOME/bin/standalone.sh
  o For Windows: JBOSS_HOME\bin\standalone.bat

Download and install/deploy dependencies:
NOTE: The following build commands assume that you have configured your Maven user settings. If you have not, you must include Maven setting arguments on the command line.
• Download and deploy the Scheduler (see Section 3.1.2)
• Download and install ui.requestCommons
  o Open a command line and navigate to the root directory of the request commons Project.
  o Type the following command to build and install:
    - mvn clean package install

Build and Deploy the request definition web application
NOTE: The following build command assumes you have configured your Maven user settings. If you have not, you must include Maven setting arguments on the command line.
• Make sure you have started the JBoss Server as described above.
• Open a command line and navigate to the root directory of the request definition Project.
• Type this command to build and deploy the archive:
  o mvn clean package jboss-as:deploy
This will deploy target/ui.requestDefinition.war to the running instance of the server.

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

Undeploy the Archive
• Make sure you have started the JBoss Server as described above.
• Open a command line and navigate to the root directory of the request definition Project.
• When you are finished testing, type this command to undeploy the archive:
  o mvn jboss-as:undeploy
3.5.1.9 Developers Guide

3.5.1.9.1 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 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.
D4.3.1 Core OpenIoT Middleware Platform

- **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.

### 3.5.1.9.2 Localization support

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

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

### 3.5.2 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 20).

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.
3.5.2.1 Deploy From Source

Start 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

Download and install/deploy dependencies

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.

- Download and deploy the scheduler and sdum (see previous sections)
- Download and install ui.requestCommons
  - Open a command line and navigate to the root directory of the request commons Project.
  - Type the following command to build and install:
    - mvn clean package install

Build and Deploy the request definition web application

NOTE: The following build command assumes you have configured your Maven user settings. If you have not, you must include Maven setting arguments on the command line.
• Make sure you have started the JBoss Server as described above.
• Open a command line and navigate to the root directory of the request presentation Project.
• Type this command to build and deploy the archive:
  o  mvn clean package jboss-as:deploy
• This will deploy target/ui.requestPresentation.war to the running instance of the server.

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

Undeploy the Archive
• Make sure you have started the JBoss Server as described above.
• Open a command line and navigate to the root directory of the request presentation Project.
• When you are finished testing, type this command to undeploy the archive:
  o  mvn jboss-as:undeploy

3.5.2.2 Developers Guide

3.5.2.2.1 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 SdumServiceResultSet 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.

3.5.2.2.2 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.
4 SOURCE CODE & BINARIES

4.1 Source Code Availability

The OpenIoT repository is hosted at the GitHub\textsuperscript{17} which can be found at the following link: \url{https://github.com/OpenIoTOrg/openiot}

The OpenIoT repository is divided in branches. The branches are divided in 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 divided in functionality themes. For example all utilities are under the “utils” folder and all user interfaces under the “ui” folder. The code structure (which is available here: \url{https://github.com/OpenIoTOrg/openiot}) is provided below:

- `doc`: provides all the related documents with the platform.
- `Modules`: provides the core modules of the platform
  - x-gsn
  - scheduler
    - scheduler.core
    - scheduler.client
  - sdum
    - sdum.core
    - sdum.client
  - lsm-light
    - lsm-light.client
    - lsm-light.server

\textsuperscript{17} \url{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
  o ui.requestDefinition
  o ui.requestPresentation
  o ide
  o RDFSensorSchemaEditor
• utils: provides utilities related with the platform
  o demoData
  o lib
  o utils.commons
  o

4.3 Binaries Availability

OpenIoT binaries are available through the OpenIoT Wiki site\(^{18}\) under the Users>Downloads\(^{19}\) section. They follow the versioning of the stable releases and are currently in the alpha version. 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. Currently a manual installation option is only available (an installation guide is provided separately for every module in Section 3) but an auto configured option will be investigated in future releases (i.e. VirtualBox\(^{20}\) with everything preinstalled).

\(^{18}\) https://github.com/OpenIotOrg/openiot/wiki
\(^{19}\) https://github.com/OpenIotOrg/openiot/wiki/Downloads
\(^{20}\) https://www.virtualbox.org/
5 PLATFORM CONTAINERS, DEVELOPMENT 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\(^{21}\). 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\(^ {22}\). 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\(^ {23}\) 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)\(^ {24}\)**: 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.

\(^{23}\) [http://www.jboss.org/resteasy](http://www.jboss.org/resteasy)
• **PrimeFaces**\(^{25}\): 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**\(^{26}\): 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**\(^{27}\): 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**\(^{28}\). 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:
  - **Sesame**\(^{29}\). 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.

\(^{25}\) [http://www.primefaces.org/](http://www.primefaces.org/)

\(^{26}\) [https://github.com/sporritt/jsplumb/](https://github.com/sporritt/jsplumb/)

\(^{27}\) [http://code.google.com/p/javamelody/](http://code.google.com/p/javamelody/)


\(^{29}\) [http://www.openrdf.org/](http://www.openrdf.org/)
• Testing Framework.
  o **JUnit**\(^{30}\): is a unit testing framework for the Java programming language.

• SPARQL Client.
  o **Sesame**\(^{31}\): For accessing SPARQL interfaces Sesame client is used.

• Logging.
  o **Logback**\(^{32}\): 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.

### 5.1 OpenIoT Components-Libraries relation

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

#### 5.1.1 Scheduler

The Scheduler libraries and containers are listed in Table 9 below.

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

\(^{30}\) [https://github.com/junit-team/junit](https://github.com/junit-team/junit)

\(^{31}\) [http://www.openrdf.org/](http://www.openrdf.org/)

\(^{32}\) [http://logback.qos.ch/](http://logback.qos.ch/)
5.1.2 Service Delivery & Utility Manager

The SD&UM’s libraries and containers are listed in Table 10 below.

Table 10. SD&UM’s libraries and containers

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

5.1.3 X-GSN

The X-GSN’s libraries and containers are listed in Table 11 below.

Table 11. X-GSN’s libraries and containers

<table>
<thead>
<tr>
<th>Container/Library Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>1.6</td>
</tr>
<tr>
<td>antlr</td>
<td>2.7.7</td>
</tr>
<tr>
<td>webcam-capture-driver-jmf</td>
<td>0.3.9</td>
</tr>
<tr>
<td>json-simple</td>
<td>1.1</td>
</tr>
<tr>
<td>h2</td>
<td>1.1.116</td>
</tr>
<tr>
<td>xstream</td>
<td>1.3.1</td>
</tr>
<tr>
<td>jts</td>
<td>1.8</td>
</tr>
<tr>
<td>commons-codec</td>
<td>1.3</td>
</tr>
<tr>
<td>commons-dbcp</td>
<td>1.4</td>
</tr>
<tr>
<td>commons-fileupload</td>
<td>1.2</td>
</tr>
<tr>
<td>commons-io</td>
<td>2.1</td>
</tr>
<tr>
<td>commons-lang</td>
<td>2.2</td>
</tr>
<tr>
<td>commons-math</td>
<td>1.2</td>
</tr>
<tr>
<td>httpunit</td>
<td>1.6.2</td>
</tr>
<tr>
<td>joda-time</td>
<td>1.6</td>
</tr>
<tr>
<td>junit</td>
<td>4.10</td>
</tr>
</tbody>
</table>
log4j 1.2.15
mysql-connector-java 5.1.26
REngine 0.6-1
Rserve 0.6-1
opencsv 1.8
stringtemplate 3.0
ant 1.7.0
axis2 1.4.1
commons-email 1.2
org.apache.commons.collections 3.2.1
httpclient 4.0.1
httpcore 4.0.1
httpcore-nio 4.0.1
httpmime 4.0.1
mina-core 1.1.7
asterisk-java 0.3
groovy-all 1.7.1
easymock 2.5.1
easymockclassextension 2.2
jetty-all-server 7.0.2.v20100331
jansi 1.10
jfreechart 1.0.14
jibx-run 1.1.6
postgis-jdbc 1.3.0
rxtx 2.1.7
rome 0.9
cos 05Nov2002
xercesImpl 2.8.1
lsm j6_2013-06-28
net.tinyos 1.x
tinyos-java 2.1
jasperreports 3.0.0
layout x
Twitter 2.x

5.1.4 LSM-Light

The LSM-Light’s libraries and containers are listed in Table 12 below.

<table>
<thead>
<tr>
<th>Container/Library Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>1.7</td>
</tr>
<tr>
<td>Ant</td>
<td></td>
</tr>
<tr>
<td>Apache Commons</td>
<td>1.8.2</td>
</tr>
<tr>
<td>Virtuoso Jena Provider</td>
<td>2.6.2</td>
</tr>
</tbody>
</table>

Copyright © 2013 OpenIoT Consortium
5.1.5 User Interface

5.1.5.1 Request Presentation

The Request Presentation’s libraries and containers are listed in Table 13 below.

Table 13. Request Presentation’s libraries and containers

<table>
<thead>
<tr>
<th>Container/Library Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>1.6+</td>
</tr>
<tr>
<td>Json</td>
<td>20090211</td>
</tr>
<tr>
<td>Primefaces</td>
<td>3.5</td>
</tr>
<tr>
<td>Primefaces-extensions</td>
<td>0.6.3</td>
</tr>
<tr>
<td>Primefaces-extensions (codemirror addon)</td>
<td>0.6.3</td>
</tr>
<tr>
<td>Apache commons-lang3</td>
<td>3.0</td>
</tr>
<tr>
<td>Jsf-api</td>
<td>2.1.20</td>
</tr>
<tr>
<td>El-api</td>
<td>2.2</td>
</tr>
<tr>
<td>Jaxb-api</td>
<td>2.1</td>
</tr>
<tr>
<td>Jaxb-impl</td>
<td>2.1</td>
</tr>
<tr>
<td>Resteasy-jaxrs</td>
<td>2.3.1.GA</td>
</tr>
<tr>
<td>Javamelody-core</td>
<td>1.45.0</td>
</tr>
<tr>
<td>OpenIoT utils.commons</td>
<td>0.0.1</td>
</tr>
<tr>
<td>OpenIoT utils.requestCommons</td>
<td>0.0.1</td>
</tr>
</tbody>
</table>
5.1.5.2 Request Definition

The Request Definition’s libraries and containers are listed in Table 14 below.

Table 14. Request Definition’s libraries and containers

<table>
<thead>
<tr>
<th>Container/Library Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>1.6+</td>
</tr>
<tr>
<td>Json</td>
<td>20090211</td>
</tr>
<tr>
<td>Primefaces</td>
<td>3.5</td>
</tr>
<tr>
<td>Primefaces-extensions</td>
<td>0.6.3</td>
</tr>
<tr>
<td>Apache commons-lang3</td>
<td>3.0</td>
</tr>
<tr>
<td>Apache commons-io</td>
<td>1.3.2</td>
</tr>
<tr>
<td>Jsf-api</td>
<td>2.1.20</td>
</tr>
<tr>
<td>El-api</td>
<td>2.2</td>
</tr>
<tr>
<td>Jaxb-api</td>
<td>2.1</td>
</tr>
<tr>
<td>Jaxb-impl</td>
<td>2.1</td>
</tr>
<tr>
<td>Resteasy-jaxrs</td>
<td>2.3.1.GA</td>
</tr>
<tr>
<td>Javamelody-core</td>
<td>1.45.0</td>
</tr>
<tr>
<td>JsPlumb</td>
<td>1.4.0</td>
</tr>
<tr>
<td>OpenIoT utils.commons</td>
<td>0.0.1</td>
</tr>
<tr>
<td>OpenIoT utils.requestCommons</td>
<td>0.0.1</td>
</tr>
</tbody>
</table>
6 CONCLUSIONS

Deliverable D4.3.1 corresponds to the first official release of the open source implementation of the OpenIoT middleware platform. The release comprises a prototype implementation of the major components comprising the OpenIoT architecture, along with accompanying documentation which is provided as part of this document. Note that the open source implementation of the above mentioned components of the release is available within the Github infrastructure of the project (https://github.com/OpenIotOrg/openiot). Hence, deliverable D4.3.1 represents a major milestone of the OpenIoT project which marks the availability of the open source infrastructure of the project.

The milestone nature of this deliverable stems from the fact that it has cleared several technical risks of the OpenIoT project, while at the same time it has validated several technical aspects of the project. First, it has validated the OpenIoT architecture, through providing a successful instantiation of the architecture using specific implementation technologies. In this way, the deliverable has alleviated key integration risks stemming from the complexity of the architecture and its components. Second, the successful completion of this deliverable has validated the cloud/IoT integration concept, which is among the core ideas of the project. Third, the deliverable has provided a core infrastructure which can serve as the basis for implementing other project functionalities, such as resource management mechanisms, utility metrics, security and privacy schemes, as well as tools for integrated development of IoT applications. These are parts of other deliverables of the project, which are expected to use the core infrastructure provided in this deliverable.

Note that the deliverable 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, 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.

While the deliverable constitutes the first official and milestone release of the open source middleware platform of the project, it does not include the complete implementation of the OpenIoT blueprint envisaged in the project. The second and final release of the deliverable will enhance the present release towards its final version. Moreover, the first milestone release of the OpenIoT middleware platform will drive the research and implementation of several other results of the project, notably results of WP4 (Cloud Services for Internet-Connected Objects) and WP5 (Management of OpenIoT Service Infrastructures).
APPENDIX I – SCHEMATA

Table 15. OSDSpec Schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" targetNamespace="urn:openiot:osdspec:xsd:1"
    xmlns:osd="urn:openiot:osdspec:xsd:1"
    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 Sensor 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:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```
<xs:element ref="osd:requestPresentation" />
<xs:element ref="prt:query-request" />
<xs:sequence>
  <xs:element maxOccurs="unbounded" minOccurs="0"
    ref="osd:dynamicAttrMaxValue" />
</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>
```xml
<xs:sequence>
  <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:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```
Figure 21. OSDSpec Schema graph
Figure 22. OSMO Schema graph

Table 16. SdumServiceResultSet Schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  targetNamespace="urn:openiot:sdum:serviceresultset:xsd:1"
  xmlns:prt="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:schema>
```
Figure 23. SdumServiceResultSet Schema graph
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <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:element maxOccurs="unbounded" minOccurs="0" ref="CoreMetaData" />
        <xs:element maxOccurs="unbounded" minOccurs="0" ref="MeasurementCapability" />
        <xs:attribute name="name" type="xs:string" />
        <xs:attribute name="id" type="xs:anyURI" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="CoreMetaData" nillable="false">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" />
      <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:attribute name="id" type="xs:anyURI" use="optional" />
        <xs:attribute fixed="" name="type" type="xs:string" />
      </xs:sequence>
    </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 24. SensorTypes Schema graph

Table 18. DescriptiveIDs Schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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 25. 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

```java
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:

```java
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:

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

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

```xml
<source ... >
   <address wrapper="x">
      <predicate key="range">100</predicate>
      <predicate key="log">0</predicate>
   </address>
</source>
```
You can access the initialization parameter named range with the following code:

```java
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:

```java
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

```java
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)\(^{33}\), and description is a text describing the field. The following examples should help you get started:

---

\(^{33}\) See `gsn.beans.DataTypes` package
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:

```java
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

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

```java
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.

The method should be implemented as below:

```java
while(isActive()) {
    // The thread should wait here until arrival of a new data notifies it
```
Webcam example

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:

```java
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:

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

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.

```java
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";
}
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:

```java
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:

```xml
<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:

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