Project no.223975

MOBESENS

Mobile Water Quality Sensor System

Instrument: *Please tick*	~~CA~~	STREP	~~IP~~	~~NOE~~

ICT-2007.6.3: ICT for environmental Management and Energy Efficiency

D4.4.1 – Overall MOBESENS grid and networking architecture

Due M24 (May 2010)
Actual submission date: June 2nd, 2010

Start date of project: June 1st, 2008 Duration:36 months

Organisation name of lead contractor for this deliverable: GET-INT

Revision 1.0

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	X

European Commission
Information Society and Media

Deliverable D4.4.1

Overall MOBESENS grid and networking architecture

Confidential

2007 - 2013

Project Number	:	223975
Project Title	:	Mobesens
Deliverable Type	:	Report

Deliverable Number	:	D4.4.1
Title of Deliverable	:	Overall MOBESENS grid and networking architecture
Nature of Deliverable	:	Report
Internal Document Number	:	D4.4.1
Contractual Delivery Date	:	May 31st 2010
Actual Delivery Date	:	May 31st 2010
Contributing WPs	:	WP4
Author(s)	:	Eric Renault (Institut Télécom) Wassim Drira (Institut Télécom) Djamal Zeghlache (Institut Télécom)

Abstract

The main objective of this document is to report the outcome of the activities on the Mobesens grid architecture to provide an easy-to-use global communication and storage space to save, archive and search for raw and processed data collected by the MOBESENS measurement system.

A brief presentation of the global organization of the different components of the MOBESENS system highlights first the role and position of the grid. This is followed by a detailed description of the storage space (in terms of organization and capabilities...), a presentation of the networking solutions and a report on the adopted visualisation approach and the resulting capabilities.

Table of contents

1 Introduction....................5
2 MOBESENS Global Architecture....................6
2.1 Functional Architecture....................6
2.2 Abstract view of the grid architecture....................7
2.3 The grid architecture....................11
2.4 Tahoe....................12
2.5 eXist-db....................13
3 MOBESENS Storage Architecture....................15
3.1 Object model....................15
3.2 OGC – Open Geospatial Consortium....................16
3.2.1 SensorML....................16
3.2.2 Observations & Measurements....................18
3.3 Information Model....................19
4 MOBESENS Storage API....................23
4.1 REST....................23
4.2 RESTful Web Service API....................23
5 MOBESENS Visualisation Architecture....................26
5.1 Visualisation....................26
5.2 ICEfaces – AJAX Push....................26
6 Summary....................29
7 Acknowledgments....................30
8 Abbreviations and acronyms....................31
9 References....................32

List of figures

Figure 1: Functional Architecture..6

Figure 2: Abstract Implementation View..10

Figure 3: Grid Architecture..11

Figure 4: Tahoe Cloud Storage System...13

Figure 5: eXist-db Internal Organization [4]......................................14

Figure 6: The Object Model...15

Figure 7: Conceptual model of processes in SensorML [2]....................17

Figure 8: ISFET Sensor Description using SensorML............................18

Figure 9: An example of measure for the ISFET sensor.........................19

Figure 10: Metalist Model..21

Figure 11: ICEfaces - Ajax Push...27

1Introduction

The two main objectives of the MOBESENS WP4 are the development of storage and processing to store, search and retrieve data like sensor characteristics, measurements and annotations of these measures, and the development of a visualisation interface to display information about the sensor network, i.e. information about sensor themselves, about the measurements or any kind of statistics (raw data, history or even more complex operations such as computing first and higher order moments), preferably in a graphical way. This led to a set of general properties:

- Transparent data fusion and data processing from the user point of view.
- Open interfaces compliant with INSPIRE directives and SEIS objectives.
- Optimization of the MOBESENS system as a whole, in prototyping and for validation.

More specifically, the following functionalities should be provided by the designed MOBESENS grid:

- Store and manage MOBESENS water quality measurements.
- Allow measurement annotations with time and location, device ID and then store, and fuse measurements with device characteristics and relevant traceability information.
- Make data available on demand for analysis.
- Allow customized retrieval, publish, subscribe and notification.
- Visualize on going measurements for analysis, monitoring and decision.

The goal of this document is to present both the specification and the design of the MOBESENS grid architecture, including the grid services and the design of internal and external interfaces.

The document is organized as follows: first, a global overview of the architecture is provided. The organisation of data and meta-data is then developed. The next chapter presents the API that has been developed so that the other modules of the MOBESENS system can use the grid. Finally, the last chapter is dedicated to the relationship between the grid and the visualisation interface.

2MOBESENS Global Architecture

This chapter aims at presenting the different levels of understanding of the MOBESENS grid. First, the functional architecture is presented to highlight the various functionalities offered by the grid. Second, the abstract view of the architecture describes how the grid shall be seen from the end-user point of view. Then, the effective implementation of the grid is described. Finally, the last two sections briefly present the two main tools that have been used to implement the grid for data storage and indexing: Tahoe on one hand and eXist-db on the other hand.

2.1Functional Architecture

The goal of the MOBESENS grid is to provide a set of functionalities to the end-users. Figure 1 presents a global overview of these services. A central part of the architecture is the data management system which aims at storing all the information relevant to the project:

–Measurements provided by sensors.

–Application-generated data which are the result of the execution of some internal and user-provided programs.

–Meta-data provided both by sensors and/or end-users to describe measurements and application-generated data.

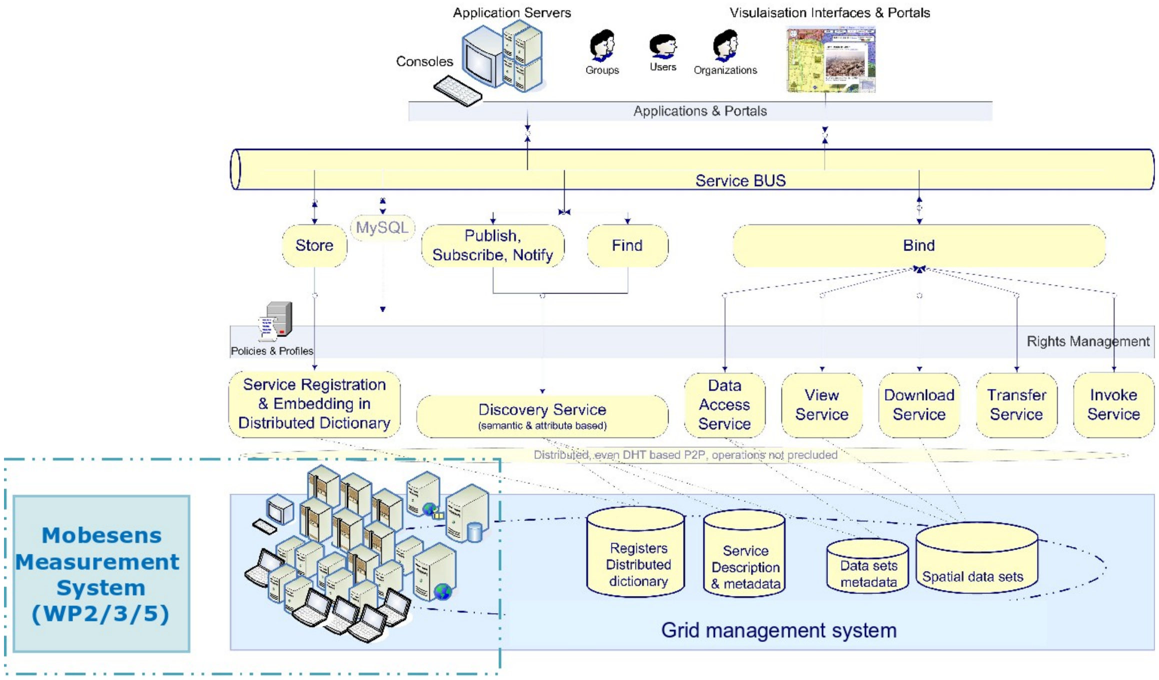

Application Servers
Visulaisation Interfaces & Portals
Consoles
Groups
Users
Organizations
Applications & Portals
Service BUS
Store
MySQL
Publish, Subscribe, Notify
Find
Bind
Policies & Profiles
Rights Management
Service Registration & Embedding in Distributed Dictionary
Discovery Service (semantic & attribute based)
Data Access Service
View Service
Download Service
Transfer Service
Invoke Service
Distributed, even DHT based P2P, operations not precluded
Mobesens Measurement System (WP2/3/5)
Registers Distributed dictionary
Service Description & metadata
Data sets metadata
Spatial data sets
Grid management system

Figure 1: Functional Architecture.

–Spatial data collected from different locations to provide an efficient visualisation interface.

This information is distributed over a set of machines using a peer-to-peer like system and are accessed and managed by a set of services that hide the complexity of the underlying implementation of the grid. Two levels of services have been identified. At the top, services accessible by end-users are providing user-friendly interfaces, while at the bottom, services are providing an abstraction of the underlying implementation. Among the services, the most noticeable ones are:

–The Store service (at the top) which is invoked by end-users to store information (data and meta-data) in the grid.

–The Publish/Subscribe/Notify service (at the top) that allows creation of and registration to events so that end-users can be notified when any kind of events arise.

–The Find service (at the top) to allow a large variety of search operations on all kind of data and meta-data stored in the grid. This service can be seen as one user-level entry point to the Discovery Service as described below.

–The Discovery Service (at the bottom) which aims at performing semantic and attribute-based searches on the different sets of data and meta-data stored in the grid system.

–The Data Access Service (at the bottom) that enables access to data, meta-data and spatial data sets.

–The View Service (at the bottom) used to render spatial data.

All the services described above are not necessarily accessible to any users as control is performed through an access rights management transversal service. This access right management is performed when accessing services at the bottom level as all top-level services are based on one or more bottom-level services and no data is directly accessible from top-level services.

Top-level services are accessible though a Service Bus on top of which applications and portals can be developed. In the scope of the MOBESENS project and more specifically in the scope of WP4, a web-based visualisation interface has been developed and improvements shall continue. However, other means to access the MOBESENS grid may be developed like for example an access from a console or UNIX terminal. This is especially interesting for the MOBESENS project as sensors are supposed to push their collected data directly to the storage grid and it would not be efficient to develop a web-based graphical client to be executed on sensors while a simple socket-based RESTful-compliant application would be lighter and efficient enough. This is the mechanism that has been chosen to allow WP2, WP3 and WP5 to communicate with the grid.

This architecture as a whole is compliant with both the INSPIRE directives and the SEIS objectives as described in [5] and [6] respectively.

2.2Abstract view of the grid architecture

The architecture we have implemented can be understood by end-users as depicted in Figure 2. The grid is mainly composed of two different spaces:

–A Storage Space which aims at storing any information in the grid. The storage is based on Tahoe [7], a secure peer-to-peer distributed storage solution, known for its efficiency and ease of management.

–An Indexation Space that compiles any meta-data provided by sensors, end-users and applications to enable efficient searches. The tool used to perform the indexation is eXist-db [8] which has been developed for the indexation of XML documents.

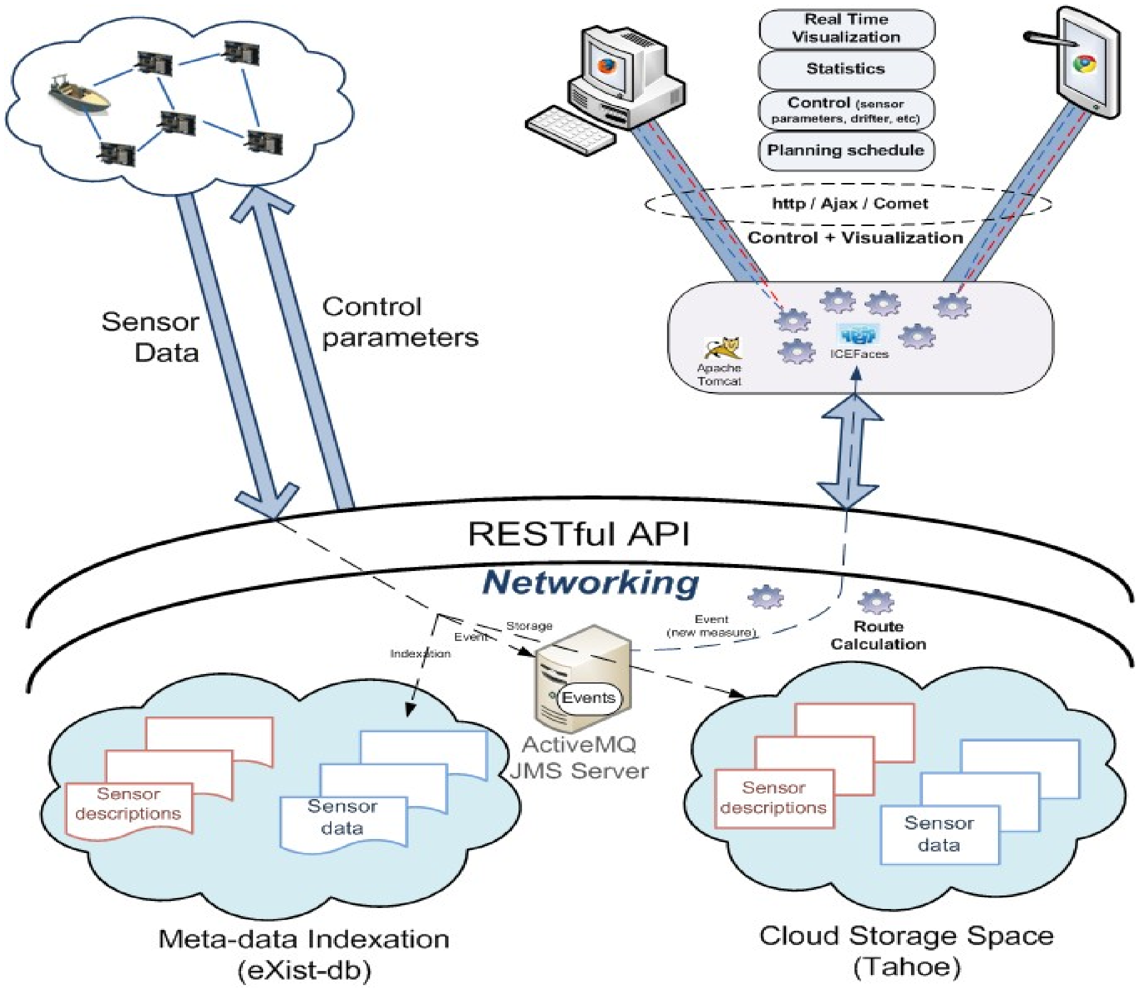


Figure 2: Abstract Implementation View.

Access to both storage and indexation spaces is performed though a RESTful compliant API as described in Chapter 4. This API allows sensors to push their collected data to the grid. These data can be visualized in a web-based client using the same API. End-users are able to display real-time and history data, perform statistics, control the sensors and send control parameters back to them, and/or plan for sensor routes.

Once stored, data in the storage space are persistent. They can be used for displaying history data or processing at any time. Another possibility consists in displaying the most up-to-date data in a real-time manner. This has been made possible with the introduction of an eventing system based on a combination of JMS (Java Message Service) Server [9] and ICEfaces [10]. This solution allows data to be visualized on the graphical user interface without refresh notification required by the user or client. Moreover, the time needed to update the graphical user interface is very small compared to the frequency of the measurements. This is a key feature since even if data are not updated in a real-time manner in the graphical user interface, the data update is performed quickly enough so that information provided to the end-user is always up to date.

Note also that all clients are synchronised and updates happen concurrently towards all the clients? Changes in the sensed or stored data trigger automatically updates in all the clients according to their subscriptions and expressed interests. Note also that when clients induce a change it is also reflected to the entire user community.

2.3The grid architecture

The implementation of the MOBESENS grid is a little bit more complex than the abstract overview. It is mainly composed of a set of sensors, servers and visualisation interfaces. Figure 3 presents the relationships between the different components of the grid.

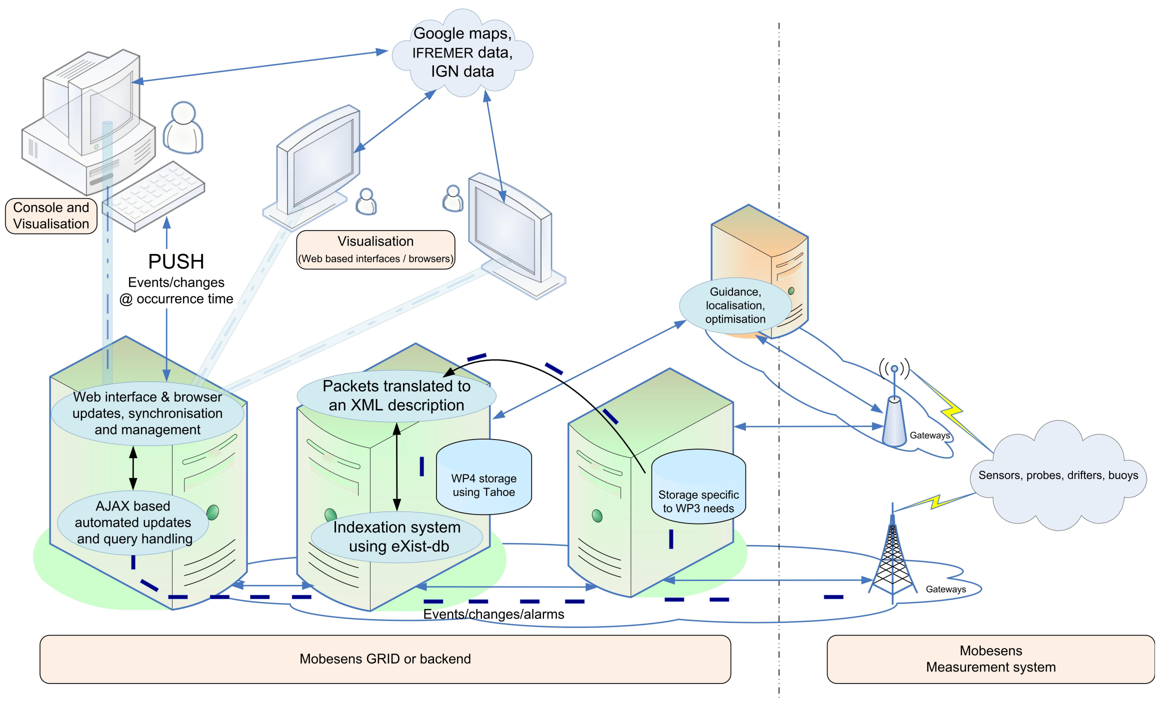

Google maps, IFREMER data, IGN data
Console and Visualisation
PUSH
Events/changes @ occurrence time
Visualisation (Web based interfaces / browsers)
Guidance, localisation, optimisation
Web interface & browser updates, synchronisation and management
AJAX based automated updates and query handling
Packets translated to an XML description
WP4 storage using Tahoe
Indexation system using eXist-db
Storage specific to WP3 needs
Gateways
Sensors, probes, drifters, buoys
Gateways
Events/changes/alarms
Mobesens GRID or backend
Mobesens Measurement system

Figure 3: Grid Architecture.

The two main components are the *MOBESENS Measurement system* which includes the sensing elements of the project (sensors, probes, drifters and buoys) and the *MOBESENS Grid* that includes both servers and visualisation interfaces. This last component is composed of:

–A set of servers to store all data for the project (the storage space) using the Tahoe filesystem.

–A server to store the indexation system (the indexation space) using the eXist-db database system.

–A server used by WP3 to store their specific data (typically, all frames provided by the WiseNodes in order to make sure no data were lost on the path from the sensors to the grid).

–Any number of visualisation interfaces that can be used for both displaying data or controlling sensors. For visualisation, geographical data can be taken from various third parties like Google Map, IFREMER and/or IGN.

–A server devoted to compute the location of sensors, optimize their path to destinations.

Updates of real-time data in the visualisation interfaces is performed using AJAX Push. This element is located in the web interface server and is connected to all visualisation interfaces.

2.4Tahoe

Tahoe-LAFS (for Least-Authority Filesystem) is an open source, secure, decentralized, fault-tolerant, peer-to-peer and cloud filesystem. The Tahoe-LAFS architecture is composed of three layers:

–At the top are applications using the filesystem.

–At the middle layer, the decentralized filesystem is organized as a directed graph in which the intermediate nodes are directories and the leaf nodes are files.

–At the bottom is located the key-value store. In this context, the keys are "capabilities" – short ASCII strings -- and the values are sequences of data bytes. A key identifies a file or a directory in the filesystem with the authority to do something with it (such as reading or modifying the contents).

Data are encrypted and distributed across a number of nodes. The storage of a file in the Tahoe filesystem follows the steps shown on figure 4:

–The file is encrypted.

–The cipher text is divided into small segments.

–An erasure coding function is applied on each segment to produce blocks of which only a subset is needed to reconstruct the segment (only 3 out of 10, with the default settings).

–One block from each segment is sent to a given server.

The set of blocks on a given server constitutes a "share". Therefore, a subset from the shares (i.e. 3 out of 10, by default) are needed to reconstruct the file. A Merkle Tree is built from the blocks to conduct correctness verification of a subset of the data without requiring all the data.

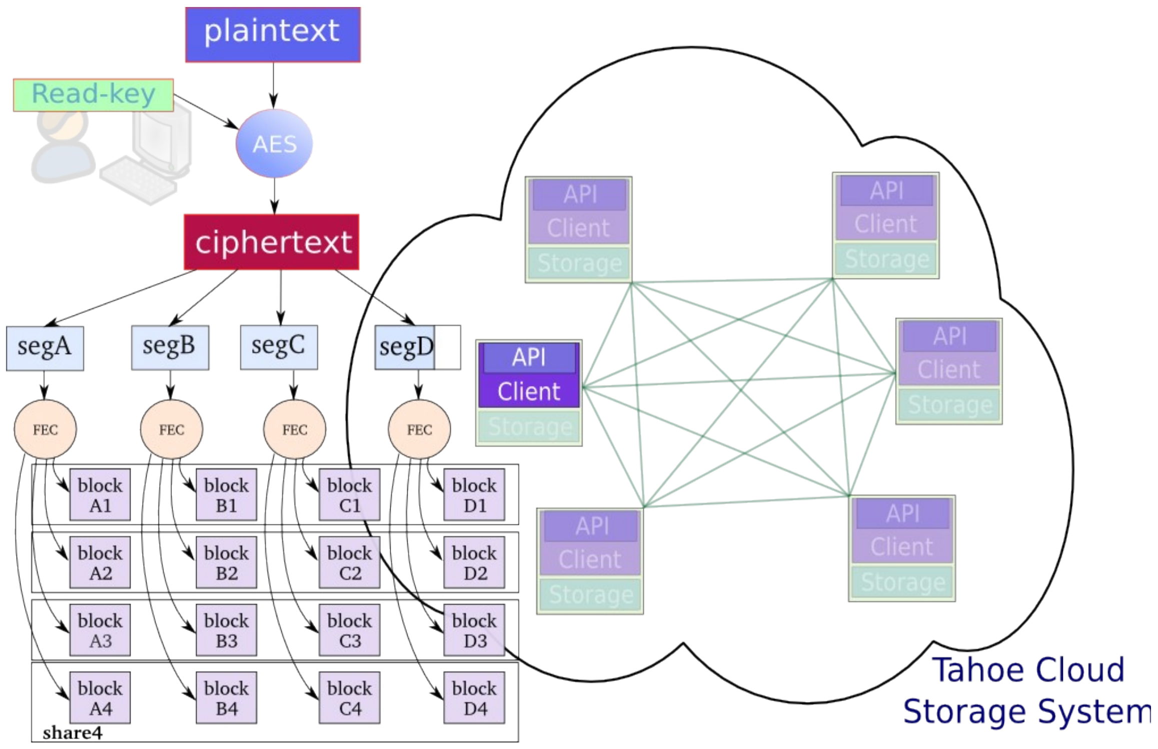


Figure 4: Tahoe Cloud Storage System.

There exists two main ways to interact with the Tahoe filesystem: the RESTful web API and the SFTP API. In the current stable version of Tahoe-LAFS (v1.6), storing and extracting files through the RESTful API has a significant overhead of around 1.2s whatever the size of the file. As a result, due to performance requirements, the SFTP API is used in the scope of the MOBESENS project.

2.5eXist-db

eXist-db is an open-source native XML database, i.e. it indexes XML documents and provides efficient execution of search operations. In order to improve the search, eXist-db also allows the creation of collections and offers the possibility to perform a search on a given collection or set of collections to improve efficiency even more. Figure 5 presents the current organization of the eXist-db framework. It highlights the fact that when an XML document or a set of XML documents are presented to eXist-db, their data are distributed over several indexes:

–**dom.dbx** is used to store the XML documents as is. This is very important to return the content of the XML document back to the users.

–**structure.dbx** creates a index based on both elements and attributes of XML documents. This index improves the efficiency of search when looking for documents in which a given element or attribute is associated a specific value.

–**collections.dbx** organizes the XML documents into an hierarchy as specified by the users. Providing a hierarchy of the XML documents allows to perform requests on a subset of the whole set of documents.

–**symbols.dbx** is the compilation of all elements, attributes, words that are used in the XML documents. It also enables name or ID mapping and resolution.

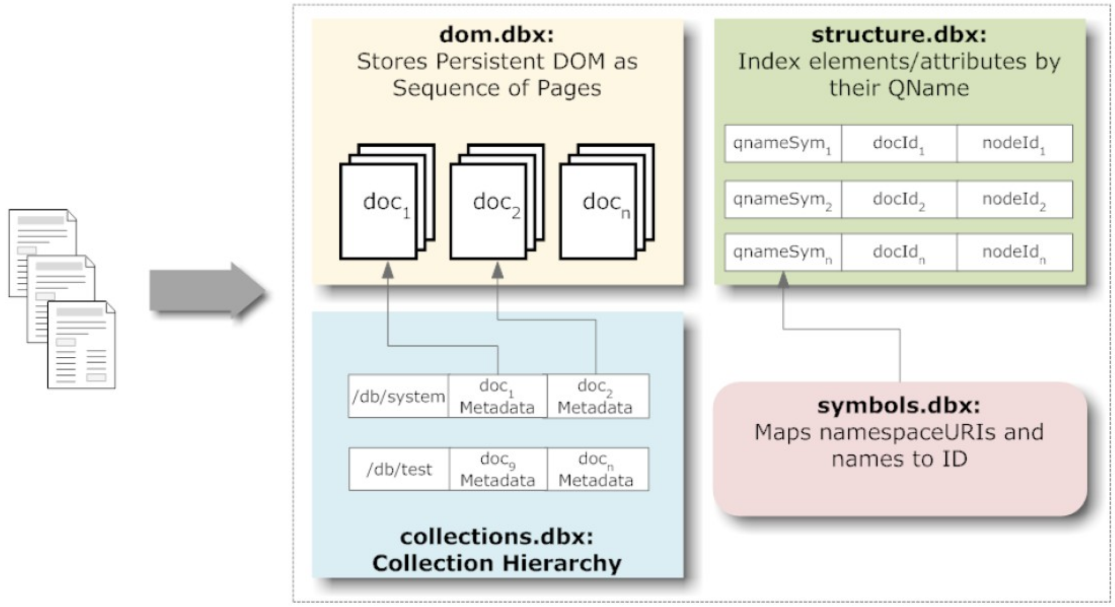


Figure 5: eXist-db Internal Organization [4].

Interactions with eXist-db may be performed through different APIs, including RESTful and XML-RPC. The one chosen in the scope of the project is XML-RPC.

As a result, various standard API may be used to send requests to eXist-db, like XML:DB, Cocoon, XML-RPC and the SOAP interface.

The expression of requests for search can be done using XQuery. As of today, eXist-db is the database system that is the most conformant to the XQuery specifications. Other languages like SQL are provided as external modules and may be used to perform requests.

3MOBESENS Storage Architecture

This chapter aims at describing the way data, measurements and meta-data are stored and organized. The first section presents the two kinds of objects that are used to store data and measurements on one hand, and meta-data on the other hand. Then, the next three sections focus on the specific storage of sensor characteristics using SensorML and sensor measured data using O&M. The last section of the chapter introduces the information model used to feed the indexation space with meta-data.

3.1Object model

Data in the storage space are divided up into two levels (as presented in Fig.6). At the bottom, BOs (for Bit-level Objects) are storing the effective raw data. At the top, IOs (for Information Objects) are storing the meta-information associated to the objects.

Bit-Level Objects. There is no specific structure for the storage of BOs. From the storage space point-of-view, a BO only consists in an array of bits that has to be stored and retrieved in the same order. As the size of BOs is fixed, more than one BO may be required to store a complete object. As a result, a BO may be understood as a block of data in the Unix file system or as a chunk in BitTorrent. Each BO is associated a cryptographic identity. This cryptographic identity is not revealed to the end-user and is only used internally by the storage space.

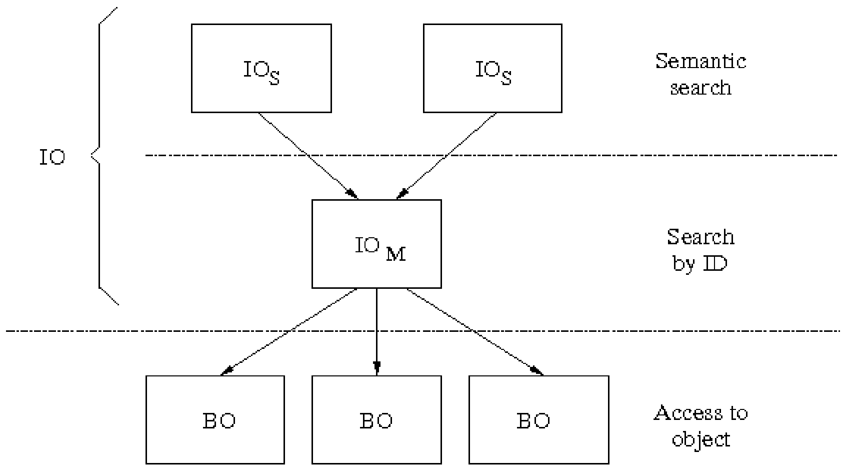


Figure 6: The Object Model.

Information Objects. There are of two different types. On one hand, IO_M are used to manage the access to BOs; on the other hand, IO_S are used to provide semantic information about objects. For the purpose of consistency, both conform to the metalist syntax as presented below. However, data they store are different:

–IO_M are typically storing data to manage the access to objects, e.g. the list of BOs that are composing the object, the set of users that are allowed to access the object and their access rights for reading, writing, etc. Even though these data are controlled by the end-users, they are managed by the grid. As there may be critical information for the system, they are not indexed and cannot be searched for by the end-users.

–IO_S are storing more general information that are provided, managed and controlled by the end-user. As long as the data provided in IO_S are conforming to the Metalists, there is no restriction on their use. As they are the typical data one will use to perform a semantic search, they are systematically indexed in the index space.

It is important to note that both IOs and BOs are stored in the storage space. This is mandatory due to the fact that any references or modifications of IOs must be possible and, as a result, a non indexed version of IOs must be resident. Therefore, IOs are all stored in the storage space using sets of BOs that are accessible through there associated IO_M.

3.2OGC – Open Geospatial Consortium

The Open Geospatial Consortium (OGC) is a non-profit, international, voluntary consensus standards organization that is leading the development of standards for geospatial and location-based services [1]. OGC has launched the Sensor Web Enablement project to enable interoperability between traditionally disparate community sensor networks. The project has been producing two interesting standard XML encodings: SensorML (for Sensor Modeling Language) and O&M (for Observations and Measurements Schema).

3.2.1SensorML

The preliminary work for the specification of SensorML began in 1998 and version 1.0 was available on July 2007. SensorML provides standard models and XML Schemas for describing the geometric, dynamic, and observational characteristics of sensors and sensor systems. It supplies information needed for the discovery of sensors and the location of sensor observations. SensorML is flexible since it allows the description of simple sensors as components and complex systems as interconnected collections of components and processes (see Fig. 7). This property is extremely useful in the scope of the MOBESENS project since the platform (the set of WiseNodes) and sensors (for example, the ISFET, the VIP system and the acoustic resonator) are provided from different partners and various combinations can be made depending on the needs.

Figure 8 shows a part of the description of the ISFET sensor which is divided into eight blocks. Each block includes specific information to be used for searching, discovering or interconnecting with the other components:

–**Keywords**, **Identification** and **Classification** properties provide both human and software readable names. Keywords give a list of words that defines the tags for the component and that can be used to enable text search on a collection of sensor descriptions. Both Identification and Classification properties provide relevant information that can be mined to support asset discovery and cataloging. Identification contains relevant information to identify the component like its name, a unique identifier and/or its manufacturer. Classification specifies the application of the component, the type of the sensor and the observed phenomenon.

–**Characteristics** and **Capabilities** properties provide information that are useful for discovery and assistance to humans. Typical Characteristics could include physical properties or power requirements, while Capabilities might include measurement characteristics or operational limits.

–**Interfaces** specify the physical connectors of the sensor to connect it to other components. For example, in the scope of the MOBESENS project, the ISFET includes an RS-485 interface that allows it to be connected as a slave to the WiseNode platform.

–**Inputs** provide the inputs of the component and **Outputs** provide the outputs of the component. For example in the case of the ISFET sensor, Inputs are the “observable properties” (pH, NH4+, etc.) and the management frame structure, while Outputs describe the structure of the output frames, like the frame that contains the measurements.

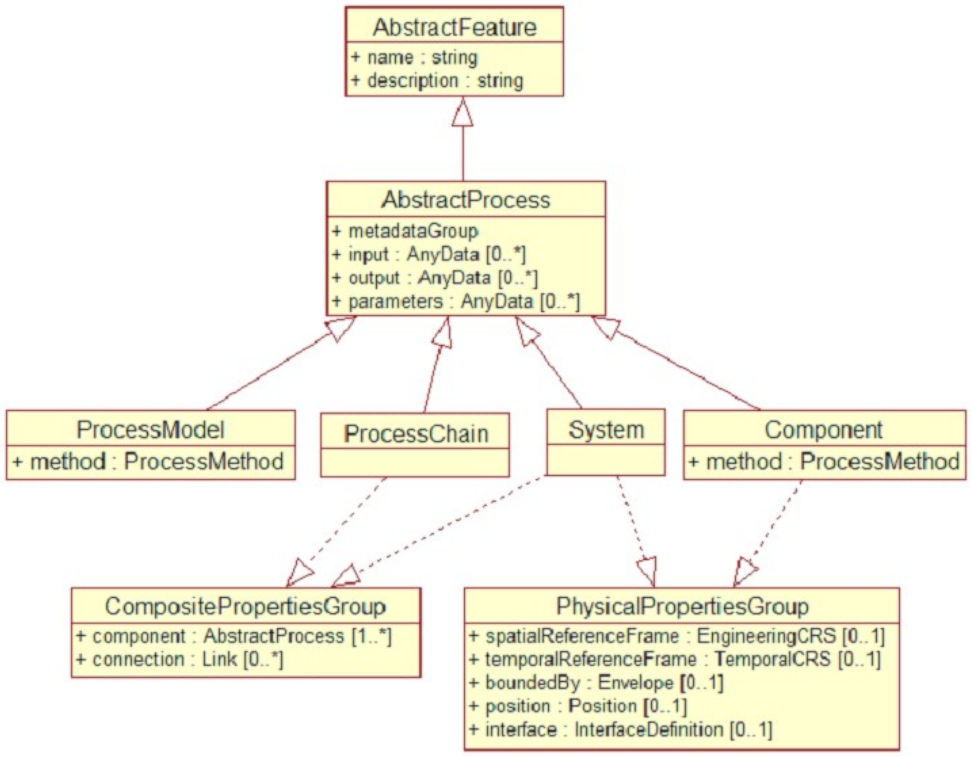
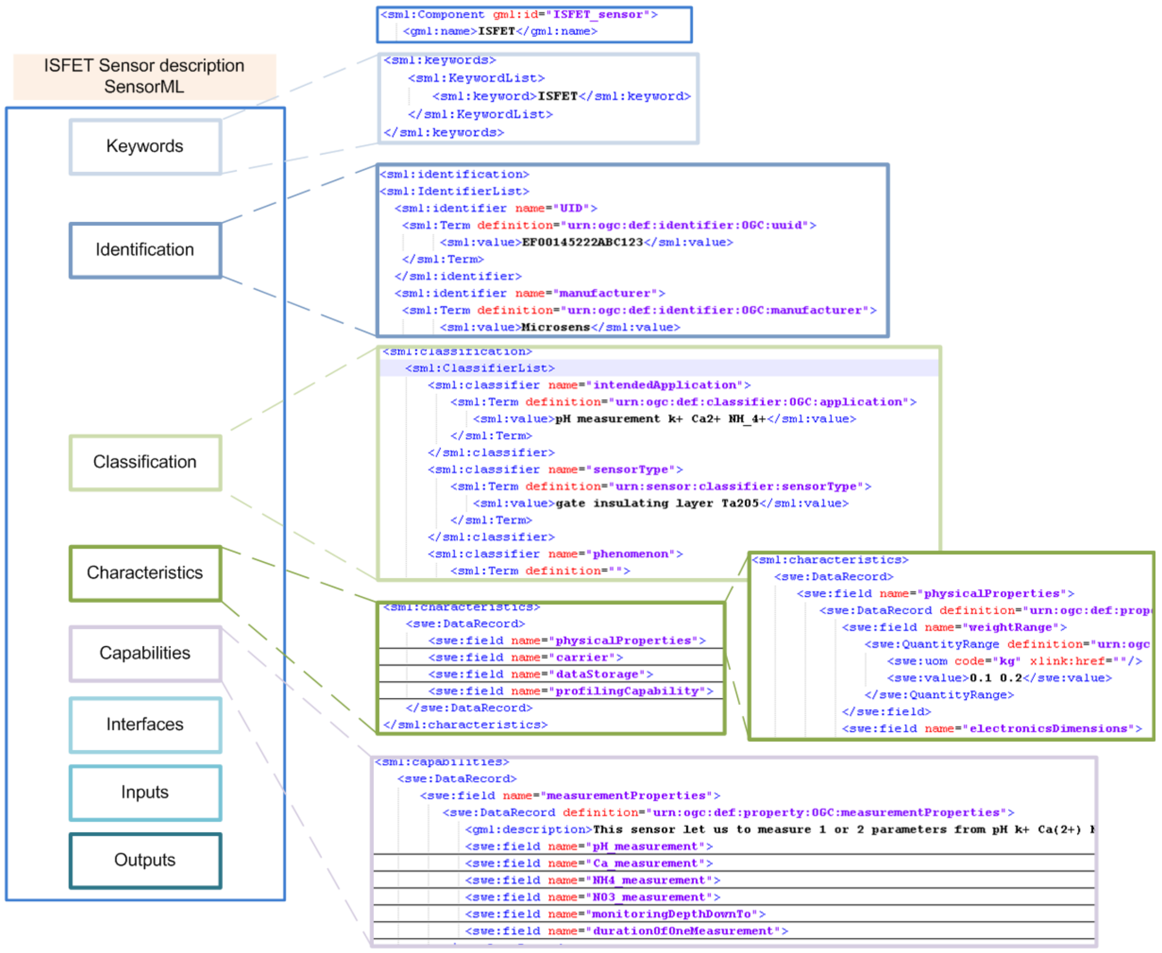

AbstractFeature
+ name : string
+ description : string
AbstractProcess
+ metadataGroup
+ input : AnyData [0..*]
+ output : AnyData [0..*]
+ parameters : AnyData [0..*]
ProcessModel
+ method : ProcessMethod
ProcessChain
System
Component
+ method : ProcessMethod
CompositePropertiesGroup
+ component : AbstractProcess [1..*]
+ connection : Link [0..*]
PhysicalPropertiesGroup
+ spatialReferenceFrame : EngineeringCRS [0..1]
+ temporalReferenceFrame : TemporalCRS [0..1]
+ boundedBy : Envelope [0..1]
+ position : Position [0..1]
+ interface : InterfaceDefinition [0..1]

Figure 7: Conceptual model of processes in SensorML [2].



```
<sml:Component gml:id="ISFET_sensor">
  <gml:name>ISFET</gml:name>
```

```
<sml:keywords>
   <sml:KeywordList>
      <sml:keyword>ISFET</sml:keyword>
   </sml:KeywordList>
</sml:keywords>
```

```
<sml:identification>
<sml:IdentifierList>
  <sml:identifier name="UID">
   <sml:Term definition="urn:ogc:def:identifier:OGC:uuid">
        <sml:value>EF00145222ABC123</sml:value>
   </sml:Term>
  </sml:identifier>
  <sml:identifier name="manufacturer">
   <sml:Term definition="urn:ogc:def:identifier:OGC:manufacturer">
        <sml:value>Microsens</sml:value>
```

```
<sml:classification>
   <sml:ClassifierList>
      <sml:classifier name="intendedApplication">
         <sml:Term definition="urn:ogc:def:classifier:OGC:application">
            <sml:value>pH measurement k+ Ca2+ NH_4+</sml:value>
         </sml:Term>
      </sml:classifier>
      <sml:classifier name="sensorType">
         <sml:Term definition="urn:sensor:classifier:sensorType">
            <sml:value>gate insulating layer Ta205</sml:value>
         </sml:Term>
      </sml:classifier>
      <sml:classifier name="phenomenon">
         <sml:Term definition="">
```

```
<sml:characteristics>
   <swe:DataRecord>
      <swe:field name="physicalProperties">
      <swe:field name="carrier">
      <swe:field name="dataStorage">
      <swe:field name="profilingCapability">
   </swe:DataRecord>
</sml:characteristics>
```

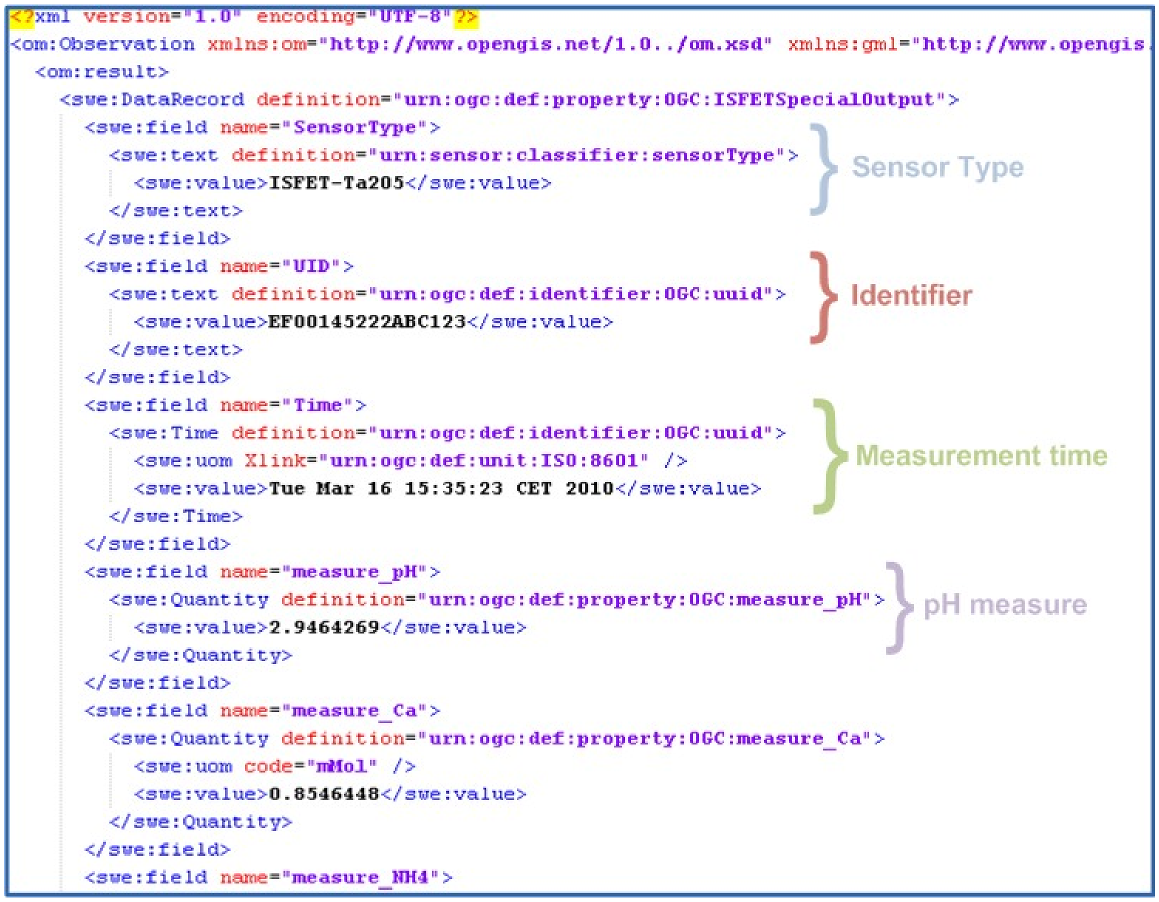
```
<sml:characteristics>
   <swe:DataRecord>
      <swe:field name="physicalProperties">
         <swe:DataRecord definition="urn:ogc:def:prop
            <swe:field name="weightRange">
               <swe:QuantityRange definition="urn:ogc
                  <swe:uom code="kg" xlink:href=""/>
                  <swe:value>0.1 0.2</swe:value>
               </swe:QuantityRange>
            </swe:field>
            <swe:field name="electronicsDimensions">
```

```
<sml:capabilities>
   <swe:DataRecord>
      <swe:field name="measurementProperties">
         <swe:DataRecord definition="urn:ogc:def:property:OGC:measurementProperties">
            <gml:description>This sensor let us to measure 1 or 2 parameters from pH k+ Ca(2+) 
            <swe:field name="pH_measurement">
            <swe:field name="Ca_measurement">
            <swe:field name="NH4_measurement">
            <swe:field name="NO3_measurement">
            <swe:field name="monitoringDepthDownTo">
            <swe:field name="durationOfOneMeasurement">
```

Figure 8: ISFET Sensor Description using SensorML.

3.2.2 Observations & Measurements

The Observations and Measurements Schema (O&M) provides standard models and XML Schema for encoding observations and measurements from a sensor, for both archive and real-time. It is a complement of SensorML since the latter is not intended to provide the framework for encoding the actual observation values. These values should be encoded and transmitted within an O&M instance. This instance should be equivalent to an output of the component or the system using its SensorML description. Figure 8 shows the measurement output of an ISFET sensor. The XML file contains different fields, each value is annotated by its name, unit of measure and a definition in a catalog or a link to an ontology. This structure allows the indexation and the storage of measurement data in both human and software readable structure. As a result, both extraction and fusion of data become easier.



```
<?xml version="1.0" encoding="UTF-8"?>
<om:Observation xmlns:om="http://www.opengis.net/1.0../om.xsd" xmlns:gml="http://www.opengis.
  <om:result>
    <swe:DataRecord definition="urn:ogc:def:property:OGC:ISFETSpecialOutput">
      <swe:field name="SensorType">
        <swe:text definition="urn:sensor:classifier:sensorType">
          <swe:value>ISFET-Ta205</swe:value>
        </swe:text>
      </swe:field>
      <swe:field name="UID">
        <swe:text definition="urn:ogc:def:identifier:OGC:uuid">
          <swe:value>EF00145222ABC123</swe:value>
        </swe:text>
      </swe:field>
      <swe:field name="Time">
        <swe:Time definition="urn:ogc:def:identifier:OGC:uuid">
          <swe:uom Xlink="urn:ogc:def:unit:ISO:8601" />
          <swe:value>Tue Mar 16 15:35:23 CET 2010</swe:value>
        </swe:Time>
      </swe:field>
      <swe:field name="measure_pH">
        <swe:Quantity definition="urn:ogc:def:property:OGC:measure_pH">
          <swe:value>2.9464269</swe:value>
        </swe:Quantity>
      </swe:field>
      <swe:field name="measure_Ca">
        <swe:Quantity definition="urn:ogc:def:property:OGC:measure_Ca">
          <swe:uom code="mMol" />
          <swe:value>0.8546448</swe:value>
        </swe:Quantity>
      </swe:field>
      <swe:field name="measure_NH4">
```

Figure 9: An example of measure for the ISFET sensor.

3.3Information Model

The metalist model as described in Fig. 10 aims at allowing end-users and the grid to provide meta-information about objects. Two pieces of information are associated to a metalist: the first one is *meta_id,* ie. the ID of the metalist itself, returned by the grid when submitting the metalist, and the second one is *object_id*, ie. the ID of the object the metalist is describing. Three ways to provide meta-information are offered to users:

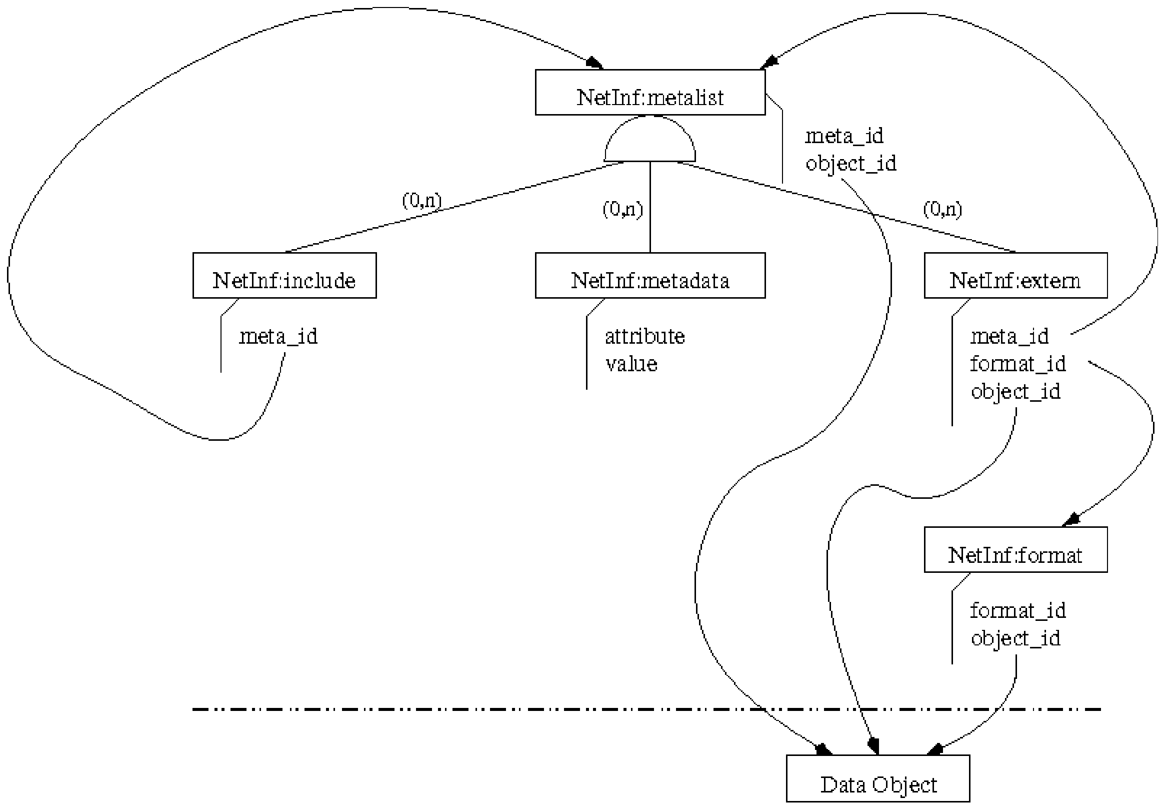


Figure 10: Metalist Model.

–*NetInf:metada* shall be used to provide meta-data directly. The *value* field contains the meta-data itself while the *attribute* field can be used to tag the meta-data. The *attribute* field is not mandatory. The *value* field is mandatory unless the *attribute* field is specified. In fact, if no *attribute* is specified, an empty *value* would result in no meta-information added to the object; however, if an *attribute* is specified, an empty *value* may be interpreted as a lack of information, or no, or false, or whatever, ie. in this case, a meta-information is effectively added to the object. Any number of meta-data may be provided inside a metalist.

–*NetInf:include* can be used to refer to another pre-existing metalist. This allows an object description to be shared by several objects. The main advantage is that it enables consistency when the same meta-information for more than one object has to be updated. The ID of the metalist to be included (*meta_id*) is the only field that has to be specified. Any number of metalists may be included inside a description using this mechanism and there is no theoretical limit on the depth of inclusion, i.e. the number of times a metalist is included inside another metalist which is also included inside another metalist, etc.

–*NetInf:extern* performs almost the same operation as the previous one, except that the meta-information to be included inside the current metalist is not supposed to be in the metalist format. As a result, the original format of the meta-data has to be specified together with the location of the meta-information so that a translation of the meta-information is performed from the original format to the metalist format. The *format_id* field is used to refer to the *NetInf:format* object that holds the process used to perform the translation. The *object_id* in the *NetInf:format* description is the ID of the object that holds the process stored in the storage space. The *object_id* in the *NetInf:extern* description is the ID of the object that holds the meta-data in the other format to be included in the metalist. If no *object_id* is specified in the *NetInf:extern* description, a *meta_id* can be alternatively provided to refer to another metalist which associated *object_id* will be used instead. If no *meta_id*, nor *object_id* is specified, the meta-information in the other format is supposed to be located inside the object itself, i.e. inside the object which ID is stored inside the inner-most *NetInf:metalist* embedding the *NetInf:extern* description. Any number of *NetInf:extern* can be included inside a metalist.

4MOBESENS Storage API

The MOBESENS Storage API is the set of functionalities that are made available from outside the grid in order to allow the different component of the MOBESENS project to store, search and retrieve their data. The first part of the chapter aims at introducing the principle of the REST architecture style. The remaining part of the chapter is then devoted to the presentation of the implemented API for the MOBESENS project.

4.1REST

REpresentational State Transfer (REST) is an architectural style for building loosely coupled systems, which means that it is not a concrete system architecture, but instead a set of constraints that are applied when designing a system architecture. The difference between an architectural style and an architecture is that the former tells general principles informing the creation of an architecture, while the latter is the fact of designing a solution to a problem according to given constraints. As a result, architectural styles inform and guide the creation of architectures [3]. The web (URI/HTTP/HTML/XML) is an instance of this style.

The REST architectural style describes the following five constraints applied to the architecture:

–Resource identification: URL, URN.

–Uniform interface: a small set of verbs applied to a large set of nouns (GET, HEAD, PUT, DELETE, POST).

–Self describing messages: they describe how to process messages (eg. MIME types).

–Hypermedia driving: if a client wants to access related resources, these should be identified in the representation returned.

–Stateless interaction: this allows moving states to clients or resources (and avoids state in server-side applications).

4.2RESTful Web Service API

A RESTful web service (or web API) is a simple web service implemented using the HTTP protocol and the principles of REST. This API was chosen for the implementation in order to guarantee the interoperability between the grid and the gateway (or any other third party applications), since almost all platforms and programming languages have implemented an HTTP stack today.

As shown below, operations supported by the grid are closely related to the REST architecture style.

Push (Data) -> ID | Error

This operation is used to push any kind of data in the storage space. This occurs when the gateway receives a new frame and wants it to be stored in the grid. The HTTP PUT function needs to be called to push the data (a file or an array of bits) on the URL. In return, it receives the unique identification of this object or an error.

A typical example of calling this function using the curl (Client URL Request Library) command line:

```
curl -T file http://157.159.103.45:8080/NetInf/api/rest
```

Publish (Metadata) -> ID | Error

This operation is used to publish a metalist. This occurs when the gateway or the operator wants to provide meta-data to describe measurements and/or sensors. The HTTP PUT function needs to be called to push the meta-data. The value returned by this operation is the unique identifier of the metalist or an error.

The calling operation in this case is very similar to the push operation described above using the curl command. In fact, the difference differs in the location of these specific data (the meta-data) in the hierarchy.

```
curl -T file.xml http://157.159.103.45:8080/NetInf/api/rest/ios
```

Get (ID) -> Data | Error

This operation is used to retrieve the content of an object from the grid using the unique identifier returned by either the push or publish operation as described above. The HTTP GET function has to be called to retrieve the content. The value returned by this operation is the effective content of the object or an error.

The call to this operation is significantly different from the two previous ones. First, no external file or data has to be provided. Second, the unique identifier the user or the application is looking for must be provided at the end of the URL.

```
curl  http://157.159.103.45:8080/NetInf/api/rest/ID
```

Search (Request) -> { Metadata } | Error

The search operation is used to retrieve information that matches a given set of criteria. This is a very useful function since it can execute queries on the indexation block, retrieve data and then build the response as described in the query. For example, a response can be an XML file, a text file or a CSV file. It can also be an error if the XQuery request is incorrect.

For example, to obtain the temperature and the depth at which the measurements were performed by sensor 99 on April 16, 2010, one can send the following query to the grid:

```
<result>
{
        for $b in collection('/db/tsp/rs2m/netinf/readings/99')//Observation
        where contains($b/Time/text(), '16/04/2010')
        return
                <rows>
                        <c>{$b/Temperature/value/text()}</c>
                        <c>{$b/Depth/value/text()}</c>
                </rows>
}
</result>
```

The call to this operation can be performed as follows:

```
curl http://157.159.103.45:8080/NetInf/api/rest/search?query=the_query
```

where *the_query* can be any XQuery expression.

Note:

1. For all the presented operations above, the IP address is dynamic and can be changed in the future or when deploying the final version. This could be done for instance using an appropriate domain name like Mobesens-grid.eu.

2. The objective of designing a RESTful API is to make the system flexible and extensible to communicate with other applications and/or project. However, in the scope of the MOBESENS project, this API is transparently used to the end-user, since users are connected to the visualisation interface to interact with the system. The gateway should use the API to push received RAW data from the wireless sensor network.

We also planed to add the support of SOS (Sensor Observation Service) to the grid. SOS is an OGC standard web service interface for requesting, filtering, and retrieving observations and sensor system information.

5MOBESENS Visualisation Architecture

This chapter describes the way the visualisation is performed using the MOBESENS architecture. The next section presents the aim of the visualisation interface. Then, Sec. 5.2 shows how the visualisation interface is automatically updated to display up-to-date data using AJAX Push. The other sections are presenting the different elements of the visualisation interface, including geographical data and data charts.

5.1Visualisation

The visualisation interface aims at displaying any information relevant to the end-user in the more convenient way. Three kinds of data have been identified:

–Geographical data. In the scope of the MOBESENS project, this is typically a map with the position of the different sensors. This map may be provided by different organizations like Google Map, IFREMER or IGN. The main advantage of being able to choose between different map providers is that some data may be available on some specific maps and not on the others.

–Real-time data. Once a sensor has collected data, this information has to be made available to the end-user as soon as possible. As the time between two successive measures may be large (up to 1 hour for the VIP for example), real-time does not necessarily mean instantaneous, but as soon as possible and without polling from the end-user. The web interface server is in charge of notifying the end-users interfaces that new data are available.

–History data and Statistics. End-users may be willing to display any set of raw data or pre-processed data for any period of time. In the same way as for real-time data, any new data in the period taken into account for the history or the statistics shall be taken into account automatically.

The most important property for all these data to be displayed in the visualisation interface is the customizability according to the end-user needs, i.e. any choice for the position or content of the different components of the visualisation interface are left to the end-user. This critical point is very important as all users are different and are not expecting the same data to be displayed on their personal visualisation interface. As a result, instead of trying to identify what would the best visualisation interface, it was decided to provide a very convenient way to let end-users chose and customize their personal environment. This also implicitly allows as many visualisation interfaces as needed by users, considering that users may use more than one visualisation environment each.

5.2ICEfaces – AJAX Push

Figure 11 shows the process of communication between the wireless sensor network and the end user visualisation interfaces, and the used technologies to maintain displayed information up-to-date in real time with minimum traffic between client and server. In the next paragraph, we describe briefly this process.

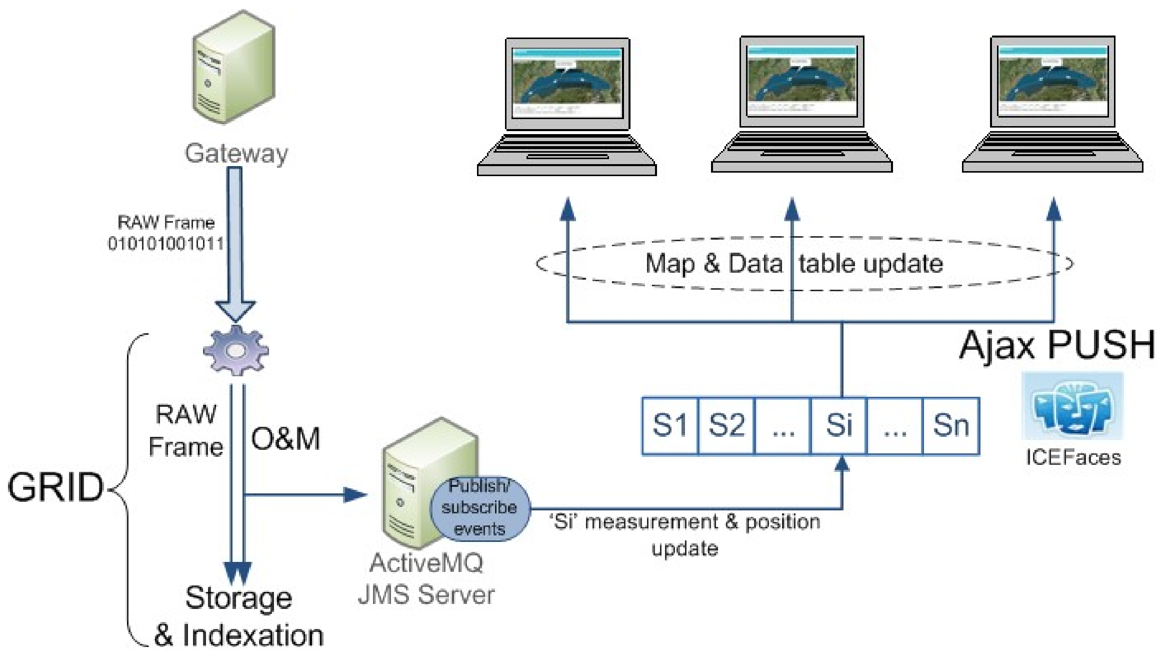


Figure 11: ICEfaces - Ajax Push.

The grid receives RAW data from the wireless sensor network through the gateway. Then, data is converted to the O&M standardized XML format. Both versions of data (raw and O&M) are stored and indexed in the system and a unique identification of the IO_M is returned to the gateway. After the data have been transformed, the O&M file is encapsulated in a message and sent to the ActiveMQ eventing server – a publish/subscribe server. The ICEfaces application and/or any other applications can subscribe to the measurement update topic.

When an update arrives, the ActiveMQ server notifies subscribers. Upon the reception of the message, the ICEfaces application executes a pseudo-code (similar to the one presented below) to notify all the subscribers and update in a real-time manner the visualisation interface using the AJAX Push technology.

```
onMessage(Message message) {
  if(message instanceof MeasurementUpdate){
    // Etract information from the message
    parsedMessage = parse(message)
    // Produce a new abstract sensorData object
    // depending on the type and readings of the sensor
    sensorData = SensorDataFactory.getInstance(parsedMessage)
    // Update the list of sensor measurements
    myDataList.update(sensorData.getUID, sensorData)
    // Notify subscribers (Visualization interface)
    SessionRender.render("SensorDataUpdateTopic")
  }
}
```

The added value of this technology is that no useless traffic is generated between the client and the server. Messages from the server to the clients are asynchronous and generated only when notifying updates is needed. This technology was chosen to update real-time sensor measurements on a map and/or in data tables. This capability is also to be added to charts visualisation to let end users compare sensors measurements in real-time through different categories of charts.

6Summary

This document presented the conception and the organization of the MOBESENS grid. It shows that the grid has been built on top of a set of tools (Tahoe, eXist-db, ICEfaces, JMS, etc.) and provides a very simple API to interact with.

At present, the MOBESENS grid has been successfully built and is operational. Any component in the project is able to push, retrieve and search for data and meta-data using the API as described in Chapter 4.

The next step consists in providing end-users a graphical interface to visualise the data stored in the storage grid. The main concern in the design and development of the end-users graphical interface is hiding the complexity of the communication with the storage grid.

7Acknowledgments

The authors of the deliverable would like to express their deep gratitude to all the people that have been involved in the development of WP4:

- Barbara Perez Felices (Institut Télécom) who participate in the development of the visualisation interface.
- Oscar Caraballo Vargas (Institut Télécom) who is in charge of the automatic translation of data from the raw format to the understandable XML format.
- Houssem Medhioub (Institut Télécom) who enabled the real-time update of data through ICEfaces.

8Abbreviations and acronyms

ActiveMQ – Active Message Queuing

AJAX – Asynchronous Javascript And Xml

API – Application Programming Interface

BO – Bit-level Object

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

INSPIRE – INfrastructure for SPatial InfoRmation in Europe

IO – Information Object

ISFET – Ion-Sensitive Field-Effect Transistor

JMS – Java Message Service

MIME – Multipurpose Internet Mail Extensions

O&M – Observations and Measurements

OGC – Open Geospatial Consortium

REST – Representational State Transfer

RPC – Remote Procedure Call

SEIS – Shared Environmental Information System

SensorML – Sensor Model Language

SFTP – Secure File Transfer Protocol

SOAP – Service Oriented Architecture

SOS – Sensor Observation Service

SQL – Structured Query Language

Tahoe-LAFS – Tahoe Least Authority FileSystem

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

URN – Uniform Resource Name

XML – eXtended Markup Language

9References

[1] The Open Geospatial Consortium. http://www.opengeospatial.org/

[2] M. Botts and A. Robin. *OpenGIS Sensor Model Language (SensorML) - Implementation Specification*, Open Geospatial Consortium, Inc.

[3] E. Wilde. *What is REST?,* WWW2009, Madrid, Spain.

[4] W. Meier. *eXist-db – Document Storage and Indexing*. XML Prague 2010.

[5] The Infrastructure for Spatial Information in Europe Directive.
http://inspire.jrc.ec.europa.eu/

[6] The Shared Environmental Information System.
http://ec.europa.eu/environment/seis/

[7] Tahoe-LAFS. http://tahoe-lafs.org/trac/tahoe-lafs

[8] eXist-db: the Open Source Native XML Database. http://exist.sourceforge.net/

[9] Java Message Service (JMS). http://java.sun.com/products/jms/

[10] ICEfaces – Open Source Ajax, J2EE Ajax, JSF Java Framework.
http://www.icefaces.org/main/home/