Proposal/Contract: 031874

CYCLOPS

CYber-Infrastructure for CiviL protection Operative ProcedureS

Technical Report on Integrating Sensors into the Grid

University of Minho

Reference: CYCLOPS_TECHREPORT_UM_SENSORS

Submission date: **December 1st 2008**

Start date of the project:	**JUNE 1st 2006**	Duration: **28 months**
Start date of this task:	**MARCH 1st 2008**	Duration: **9 months**
Partners:	University of Minho	
Editor(s):	António Joaquim André Esteves	

DOCUMENT HISTORY

Issue.revision	Date	Description of change	Author	Affiliation
1.0	25/11/2008	Main text	António Esteves Joaquim Macedo António Pina Vítor Sá Marco Caldas Luiz Lopes Nuno Lebreiro	UMinho

TABLE OF CONTENTS

1 Introduction

Following the period of integration in the CYCLOPS project [1], the University of Minho clearly identified a number of initiatives to establish stronger collaboration with other partners, related EU projects (such as DORII [2]) and initiatives, such as, the Portuguese National Grid Initiative, and engage actively with them to promote and enhance CYCLOPS results.

The CYCLOPS WP4 Technical Meeting, which took place on the 20th-21st of December 2007, on the premises of the University of Florence, in Prato, focused on the following points:

- WP4 workplan:
  - Implementation of Geospatial service layer on top of *gLite*;
  - Porting of Civil Protection applications.
- New Technical Annex and amendment:
  - Definition of the tasks of the new technical partners (EMA and UMINHO).

The initial part of the discussion involved all the participants in an in-depth analysis of issues such as:

- The need to clarify the advantages in using the Grid environment for CP applications concerning the two use-cases considered;
- General approaches to the parallelization of computing tasks for CP applications.

The final part involved small groups which focused on specific technical aspects:

- How to port SPCGD on *gLite* using the same approach planned for RISICO, considering the decision to use Grid-enabled OGC services in the SPCGD framework;
- The problem of data formats considering the possibility of porting input and output data to a common format (geoTIFF, netCDF);
- Possible approaches on the implementation of sensor access services.

After the discussion several tasks were proposed, and in particular the University of Minho was committed to:

- Enabling a working node in the CYCLOPS VO;
- Working on access to grid-enabled sensors.

This document reports the work carried out in order to fulfill the second commitment of the University of Minho: to enable access to sensors through the Grid.

2 Integrating Sensors into the Grid in order to Feed Civil Protection Applications

2.1 Introduction to Wireless Sensor Networks

Wireless networks are being used in such different areas as the military, tourism, education, stock control, discovery of ecological disasters, and medical emergencies, among others. In the context of wireless mobile computation, the user has continuous access to information through a network of communication without wire. This type of network is appropriate for situations where an installation with wire is not possible but immediate access to information is required.

A **sensor network** can be defined by means of different aspects. One possible definition for a sensor network is that of a network without wires formed by a great number of immovable small sensors planted in an ad hoc base to detect and to transmit physical characteristics of the environment, known as phenomena. The information contained in the sensors is added to a central data base.

Sensors can be used to monitor environments with difficult or dangerous access, such as the ocean depths, volcanic activities, enemy territory, forests, ambient disaster areas and places of nuclear activity. They can also be used for interactive tasks, such as finding and detonating enemy mines, searching for survivors of natural disasters or containing and isolating spilled oil to protect the maritime coast.

In this way, through the efficient use of sensor networks, we can construct concrete applications to aid Civil Protection (**CP**).

2.2 Simulation/Emulation in the Context of Integrating Sensor Networks with Grid Civil Protection Applications

With the growing popularity of the wireless world and the development of cheaper, smaller embedded processors, sensor networks have grown as an important field.

In the context of integrating sensor networks with grid civil protection applications, simulation is relevant due to the fact that (i) it can replace real sensors with a simulated version, which makes it easier, faster and cheaper to test applications, (ii) it makes it possible to scale a real network and test applications with a larger input data set, (iii) it aids in the choice of topology and localization of the network, based on the type of soil and other geographic criteria.

2.3 Emulators/Simulators comparison

In the present task, several network simulation/emulation environments were analyzed: GloMoSim, OPNET [3], TOSSIM [4], OMNet [5], EmStar, SensorSim [6], ns-2 [7], SENSE [8], ATEMU [9], SENS [10], DISENS, AVRORA [11], as well as many others. However, because of the unique aspects and limitations of sensor networks, the existing network models may not lead to a complete demonstration of all that is happening.

The comparison of simulators, done in the context of the CYCLOPS project, focused on the scalability, the popularity (existence of good references and usages), the GUI, the platforms where the simulator runs and the utilization license, among other aspects.

The tables in Appendix I show a comparison for several network simulators / emulators and focuses on aspects that are relevant to our problem.

2.4 Extending Sensor Networks with TOSSIM

TinyOS [12] is the de-facto standard and a very mature Operating System for wireless sensor networks, which consists of a rich set of software components developed in *nesC* language, ranging from application level routing logic to low-level network stack. TinyOS provides a set of Java tools in order to communicate with sensor networks via a program called *SerialForwarder*. *SerialForwarder* runs as a server on the host machine and forwards the packets received from sensor networks to the local network. Once the *SerialForwarder* program is running, the software located on the host machine can parse the raw packet and process the desired information.

Our choice for simulation environment was TOSSIM [4], an accurate and scalable simulator of entire TinyOS applications. The reasons for the choice of TOSSIM were: it is widely used; it is GPL;, it is an emulator instead of simulator, and the same code can be used on the emulation and the real hardware without any modifications.

By exploiting the sensor network domain and TinyOS' design, TOSSIM can capture network behavior at high fidelity while scaling to thousands of nodes, and by using a probabilistic bit error model for the network, TOSSIM remains simple and efficient, but expressive enough to capture a wide range of network interactions. TOSSIM was designed for TinyOS applications running on MICA2 motes, but it can be adapted to other type of motes. We can define a **mote** as being a sensors/radio/processor module used to build wireless sensor networks (WSN).

The four key elements required for a good TinyOS simulator [4], and those whose development followed, are:

Scalability: The simulator must be able to handle large networks of thousands of nodes in a wide range of configurations. The largest TinyOS sensor network that has ever been deployed was approximately 850 nodes. The simulator must be able to handle this and the much larger systems of the future.

Completeness: The simulator must cover as many system interactions as possible, accurately capturing behavior at a wide range of levels. Algorithm and network protocol simulations are helpful, but the reactive nature of sensor networks requires simulating complete applications.

Fidelity: The simulator must capture the behavior of the network at a fine grain. Capturing subtle timing interactions on a mote and between motes is important both for evaluation and testing. The simulator must reveal unanticipated interactions, not just those a developer suspects.

Bridging: The simulator must bridge the gap between algorithm and implementation, allowing developers to test and verify the code that will run on real hardware. Often, algorithms are sound but their implementations are not.

As individual motes have limited storage and CPU resources, TOSSIM can simulate accurately several motes at the same time. The TinyOS' event-driven execution (and its component-based architecture) maps well to a discrete event simulation like TOSSIM, requiring a very simple simulation engine.

The TOSSIM architecture [4] is composed of five parts: the support for compiling TinyOS component graphs into the simulation infrastructure, a discrete event queue, a small number of re-implemented TinyOS hardware abstraction components, the mechanisms for extensible radio and ADC models, and the communication services for external programs to interact with a simulation. Figure 1 shows the TOSSIM architecture.

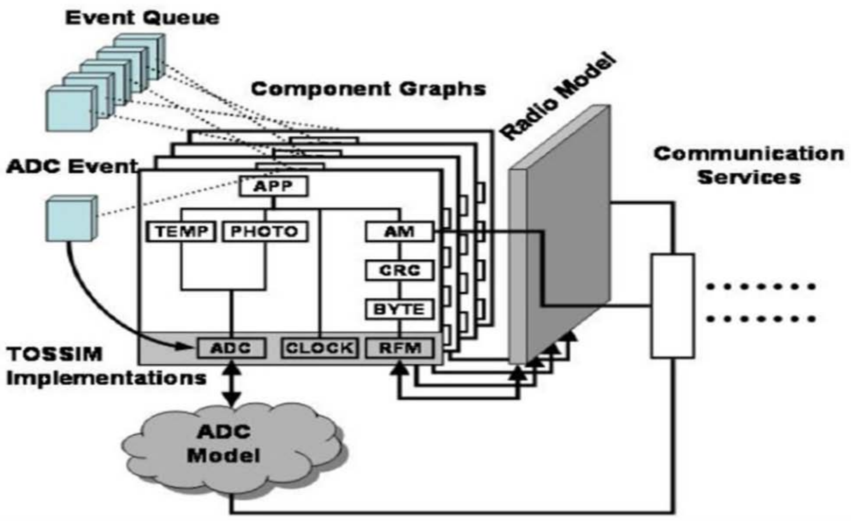

Figure 1 – The TOSSIM simulator architecture.

3 Case Study: Implementation of the Access to a Imote2 based Sensor Network with GRIDCC/DORII

The main goal of the CYCLOPS project was to join together the rid and GMES communities in order to fully exploit the Grid capabilities for GMES applications.
One of the scenarios of GMES was the existence of **r**emote **w**ireless **s**ensor **n**etworks (**RWSN**), which could monitor and transmit data about remote places in real time, such as temperature, humidity or wind speed, in order to prevent or react more accurately to situations of natural disasters, such as floods, fires or earthquakes.
In the present task, we are studying the possibilities of using a middleware called GRIDCC [13] to control, monitor and collect data from these RWSN.

3.1 The GRIDCC/DORII Middleware

The GRIDCC middleware was developed during the last three years as part of a European project, rightly called ***G****rid enabled* ***R****emote* ***I****nstrumentation with* ***D****istributed* ***C****ontrol and* ***C****omputation* (**GRIDCC**), the goal of which was to build a geographically distributed system that is able to remotely control and monitor complex instrumentation, ranging over a large number of diverse environments, from a set of sensors used by geophysical stations monitoring the state of the earth to a network of small power generators supplying the European power grid. There is now an updated version of this software resulting from another project – **DORII**, which continues the work carried out in GRIDCC.

As this middleware appears to be suitable for YCLOPS, we are engaged in an attempt to validate the tool for use in CYCLOPS, aiming at real time data collection for deployment to GRID running simulations, in the area of civil protection.

GRIDCC/DORII provides well-proven technology that can be deployed on top of existing Grid middleware, extending the grid e-infrastructure to the control and monitoring of remote instrumentation. EGEE ***gLite*** is the natural reference Grid middleware for GRIDCC and the EGEE e-infrastructure is the natural framework in which to deploy and integrate the instrument with Grid technology.

The core and novel element of the GRIDCC/DORII middleware is the ***Instrument Element*** (**IE**), which offers a standard Web service interface to integrate scientific and general purpose instruments and sensors within the Grid. The second key component of GRIDCC/DORII is the ***Virtual Control Room*** (**VCR**), which has been introduced to provide remote users with a virtual area from where they can control and monitor the instrumentation and where they can collaborate with each other, even if located in different physical sites. The third main component of GRIDCC/DORII is the Execution Service, which provides a workflow engine able to handle BPEL workflows interacting both with the new features of GRIDCC/DORII and with traditional computational and storage Grid services.
The users required that the data produced by the instrumentation was monitored and logged. Data loggers are thus foreseen both locally to the IE (files) and remotely through the built-in **SRM** (grid **S**torage **R**esource **M**anager) of the IE, which enables worldwide Grid publication of the data. Worldwide on-line monitoring is provided with a new and efficient publish/subscribe middleware developed by GRIDCC/DORII and compliant with the **J**ava **M**essaging **S**ystem (**JMS**).
Figure 2 depicts the relationship between the GRIDCC/DORII components (IE, VCR, Execution Service, Pub/Sub system) and the other Grid components.

Users interact with the remote instruments through the Virtual Control Room (VCR) which provides a prompt and highly interactive environment to control and monitor the instrumentation. Moreover, the VCR provides the users with a cooperative environment (chat, videoconference, electronic log book) to facilitate remote interactions among different operators of the instrumentation.

The Virtual Control Room (VCR) is the application scientists' interface into the GRIDCC system. The VCR is a Web-based groupware which provides collaboration support tools and the foundation to allow teams of people to control and manage Grid resources (e.g., job and workflow submission, credential management, file transfer), including remote instrumentation. In particular, the modular and layered architecture supports the common requirements of all pilot applications providing, at the same time, the possibility to easily extend the VCR functionalities with a set of more application-specific plugins. The VCR is also a development tool for GRIDCC applications via the workflows management environment and *CoGRIDCC* scripting environment.

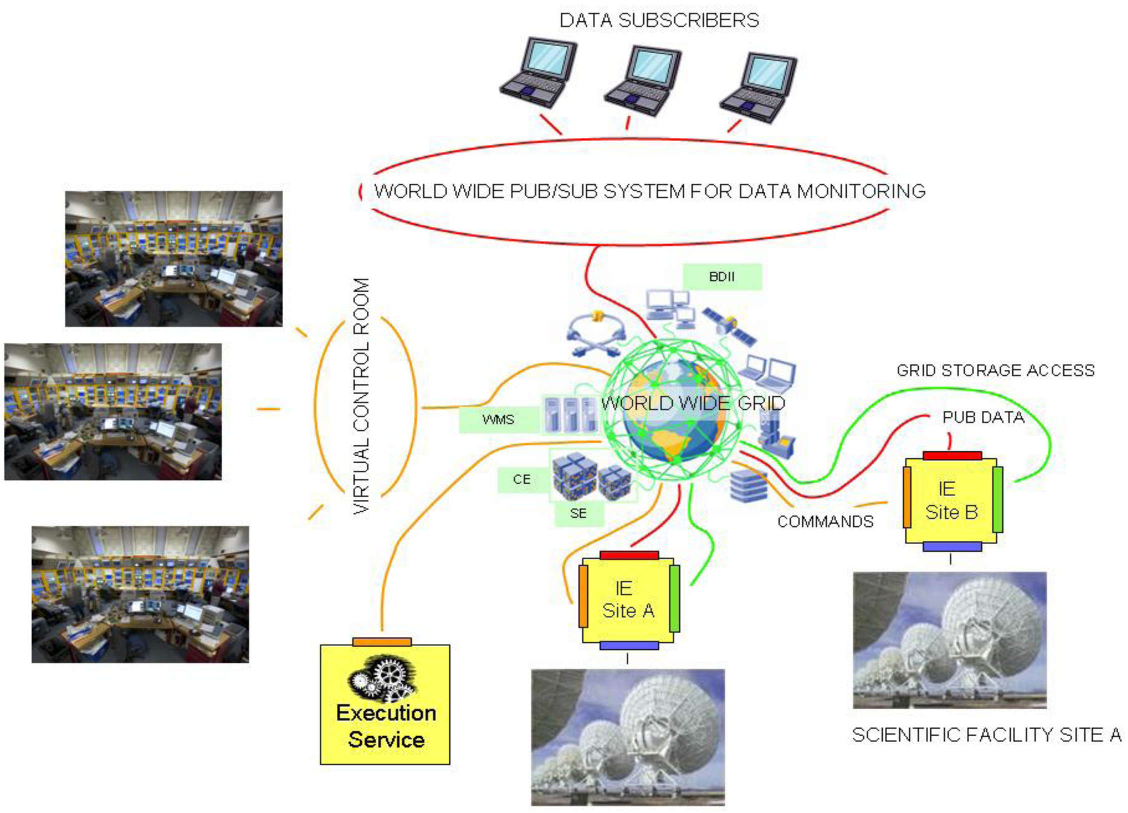

Figure 2 - Integration of instruments into the Grid.

GRIDCC is already used in at least 2 applications related to the theme of civil protection:

- A Meteorology application, which provides weather forecasts using:
    - The Virtual Control Room. The VCR is the interface through which it is possible to discover resources, execute the meteorology model, monitor its execution and retrieve the results.
    - The Skiron/Eta model, installed and appropriately configured at a HELLASGRID cluster at IASA's site.
    - GRIDCC Virtual Organization. The resources that belong to the GRIDCC VO support MPI execution in the GRID environment, which is a prerequisite of the meteorology Skiron/Eta forecasting model.

- Geo-hazards monitoring:
    - The integration of a sensors network, planning a possible technology and developing applications on behalf of GRIDCC improved middleware; all on geophysical techniques and geo-hazard monitoring.
    - According to the use case and to the logical model of geo-network activity, a Java management library has been developed to run control (remote run control) and to

maintain the Data-Logger Keithley Instruments Model 2701 that collects all data from sensors.

3.2 Description of the Imote2 Platform

Our case study for the integration of sensors into the Grid is an instrument based on Imote2 sensor modules [14], which can be used to build a wireless sensor network. The main features of an *Imote2* module is an Intel XScale PXA271 processor and a radio processor based on the 802.15.4 standard (see figure 3). The PXA271 processor can be configured to work with low voltage (0.85V) and very low frequency, which will allow a lot of power to be saved, or to be configured to work with a much higher frequency, consequently consuming much more power. The third main feature of the Imote2 is the possibility of being connected with a sensor board ITS400 that includes a 3D accelerometer and temperature, humidity and light sensors (see figure 4).

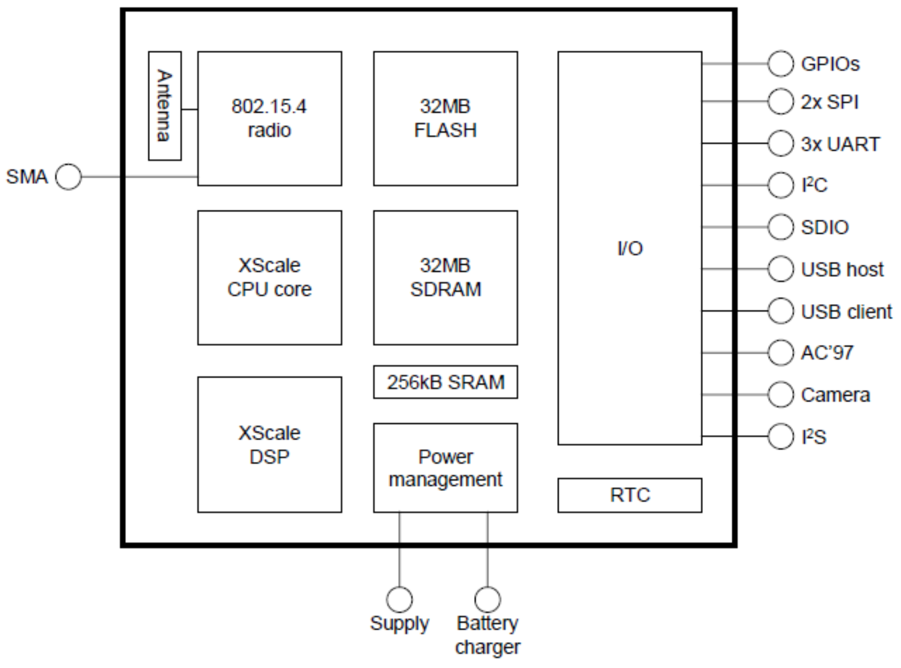

Figure 3 - Imote2 radio/processor board block diagram.

After studying the alternatives for the WSN operating system, TinyOS 1.x was selected. The reasons for this selection were:

- TinyOS is the de-facto standard and it is a very mature Operating System for wireless sensor networks;
- TinyOS consists of a rich set of software components (such as active messaging, RF communication, temperature/light/ADC sensor access and clocking) developed in nesC language.

nesC will be used to develop a customized application for sensor data acquisition, and data forwarding via wireless (802.15.4) to a base station. This base station is connected to a PC through a serial (USB) port. A customized java application will be developed to communicate with the Instrument Manager, for remote management of the sensor network.

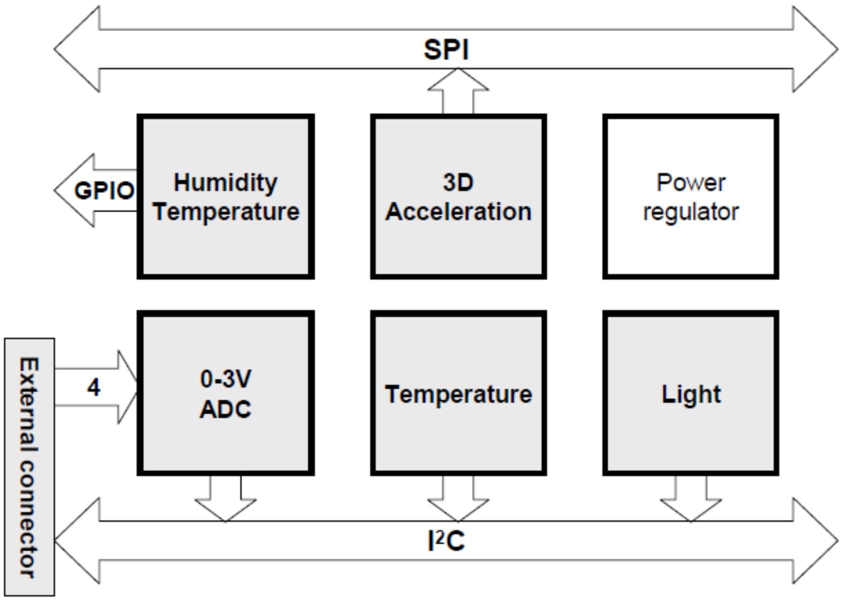

Figure 4 - Imote2 sensor board block diagram.

3.3 Implementation of the IE/IM Following the GRIDCC/DORII Approach

The architecture of the system proposed to connect an Imote2 based WSN instrument into the Grid is synthesized in figure 5. The main components of this architecture are:

- The wireless sensor network itself and its base station;
- The instrument access services host with the CJI, observations database, SOS, and other related services;
- The host of the Instrument Element, which may contain several Instrument Managers and associated custom plugins to interface with CJI;
- The VCR host that allows users to access the instrument.

Each remote Imote2 module, with associated sensors, will run a customized nesC application that monitors temperature, light, humidity and 3D acceleration. All Imote2 nodes of the network are connected to an Imote2 base-station through RF. The Imote2 base-station is serially (USB) connected to a laptop, the main task of the application it runs being to perform packet routing between the remote Imote2 and the laptop. The laptop (**I**nstrument **A**ccess **S**ervices **H**ost) will have a Custom Java interface for sensor operations and TCP/IP connection to the correspondent IM. The IASH also hosts the Observations Database and the OGC services (SOS, SPS, ...). The remaining components of the architecture are the VCR and IE hosts, which in our implementation are installed in different desktops, both with Grid connection. There is a single IE for the instrument (WSN), but several VCR users may connect to the IE at the same time. The implementation of the IM, for an Imote2 sensor, includes a custom interface to communicate with the CJI. The main software components of the architecture are presented in figure 6.

In the present task of the CYCLOPS project several remote sensor networks need to be monitored, which provide data to civil protection Grid applications. These applications require real time data collected from those networks in order to help civil protection authorities to make decisions based on simulations. The available middleware (*gLite*) supports computation and storage tasks running over the Grid, but it does not allow the control of remote instruments. An evaluation of GRIDCC/DORII middleware, an extension of *gLite*, shows that it seems to be a suitable solution in order to access and control remote instruments.

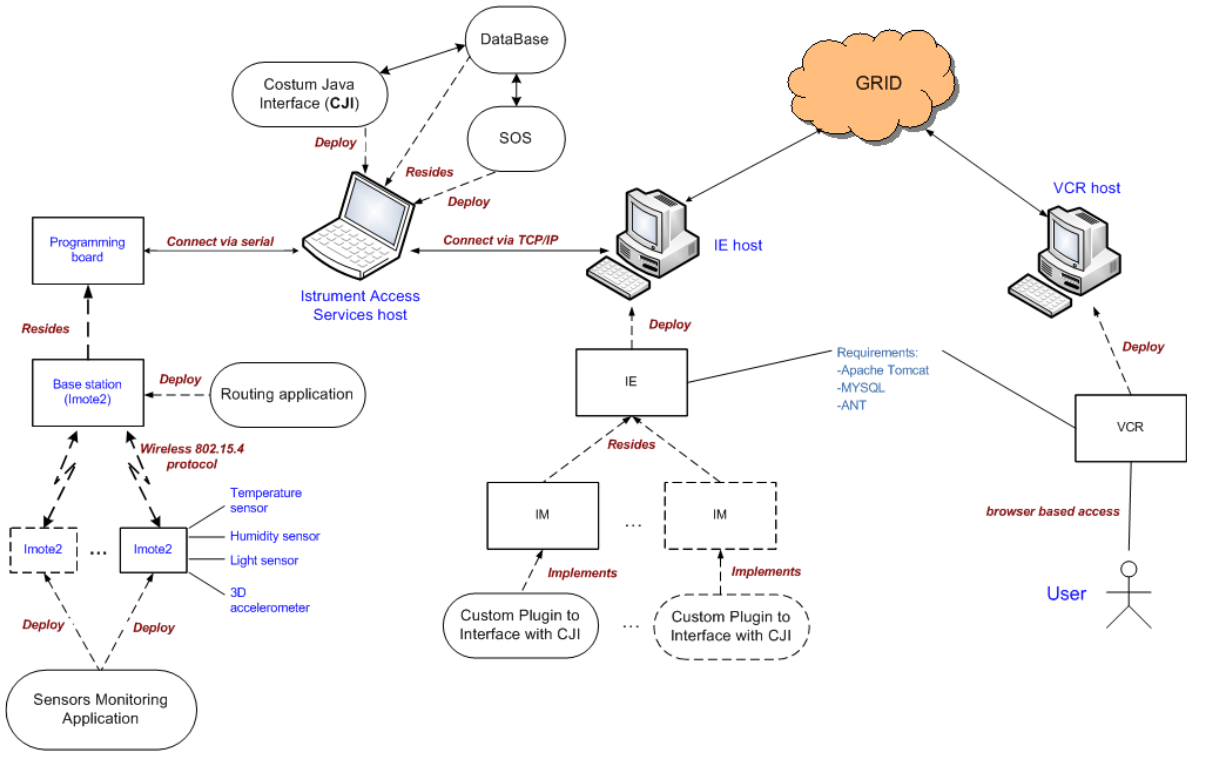

Figure 5 – Proposed architecture to integrate a WSN instrument into the GRID.

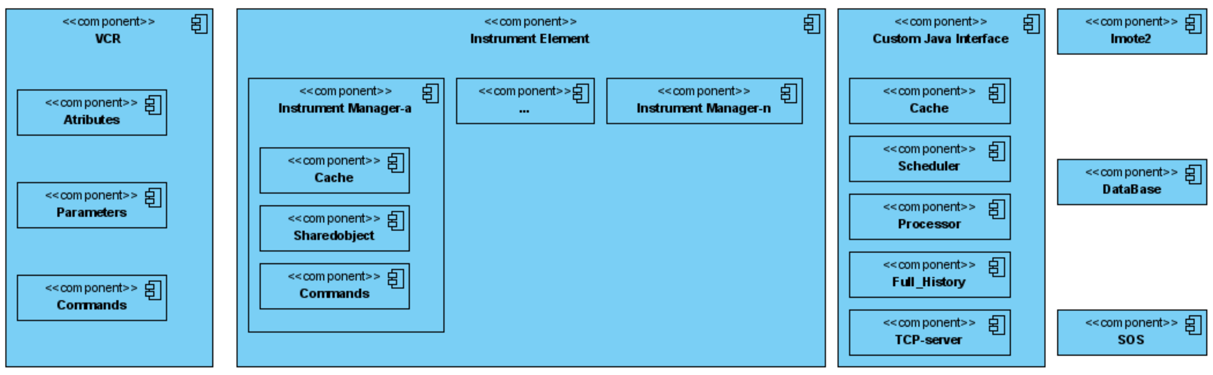

Figure 6 – The main software components of the architecture.

So, there is an *Internet portal*, the VCR, where scientists or civil protection people may log in and see the available RWSN, choose the one they are interested in and monitor or interact with it, using custom defined commands. This VCR allows them to save the data, share it, or send it to grid-running simulations with a user-friendly graphical interface.

A RWSN must be virtualized in order to “appear” on the VCR so the users can monitor and interact with it. This virtualization is done by an Instrument Element that will integrate the specifications and capabilities of the RWSN using one or more Instrument Managers. The Instrument Element (IE) is basically a container for Instrument Managers.

Our prototype system uses one IM for each sensor, due to GRIDCC/DORII software constraints, as our instrument – the wireless sensor network, is a very particular one and interaction with (use of) the sensors has singular needs, namely regarding time management and duration of the jobs. Another particularity that brings complexity to the case is that the real instrument is the single sensor – not the network. Maybe the network itself could have been virtualized as a single instrument, but this would have resulted in a large virtual instrument with a huge number of parameters, the result of which would not have been friendly at all.

As consequence of the constraints imposed by the DORII middleware, our instrument must operate in one of two modes (figure 7): *real time mode* or *user interaction mode*. When there are no users sending commands to the instrument, the instrument performs stand alone monitoring and the users automatically watch the real time observations. In this mode, the users can only handle real time observations: save, disseminate to other users or send to Grid application (figure 8). When the user wants to submit a command, to request data or schedule observations, the instrument operates in user interaction mode (figure 9).

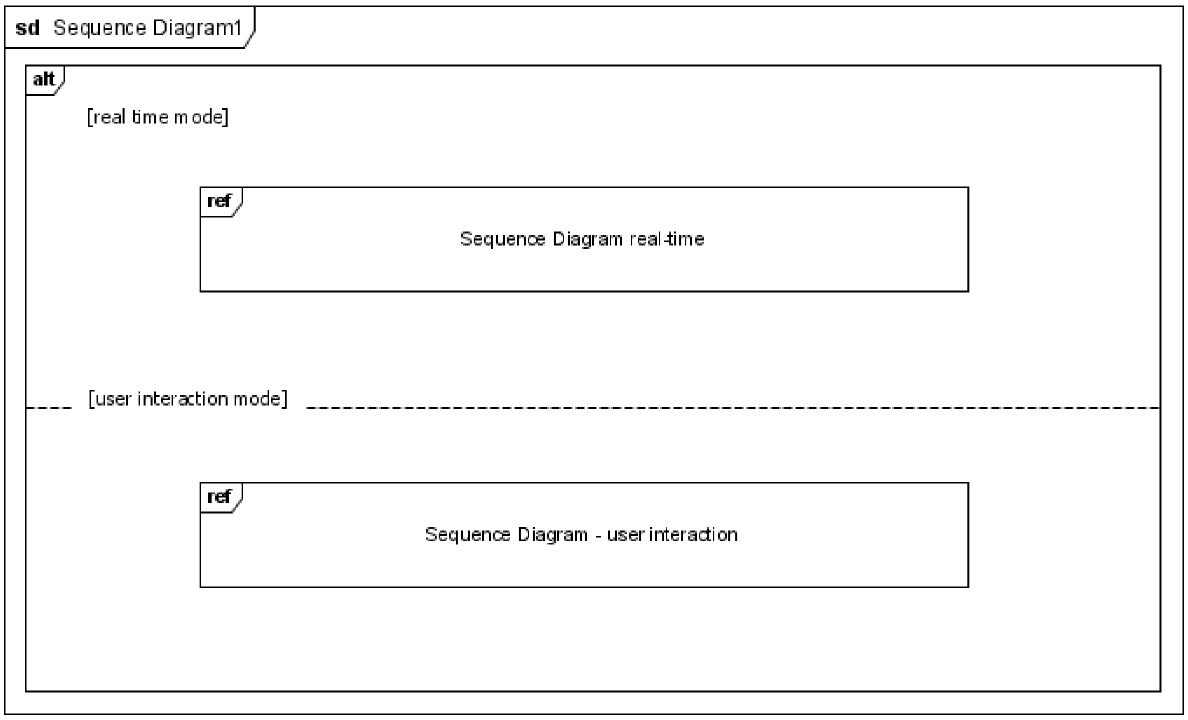

Figure 7 – Top level UML sequence diagram used to explain the instrument operation modes.

The last block of the system includes the real sensors (the whole network) and their connection to the IE. A **Custom Java Interface** (**CJI**) has been developed to perform the "translation" of the commands and data between the IM and the sensors and to manage the connection itself using TCP/IP.

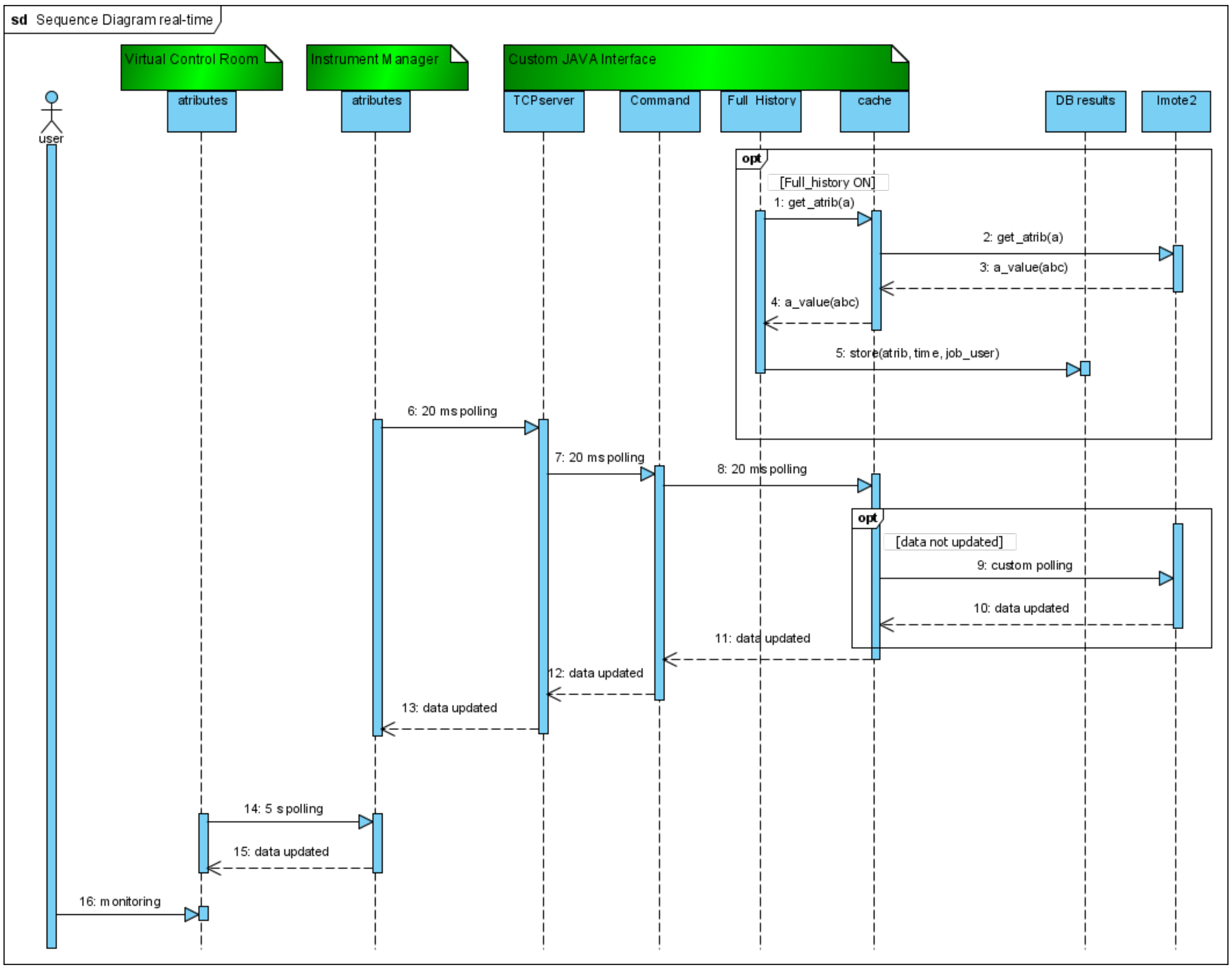

Figure 8 – Real time mode timing.

This CJI gains some major features in the prototype, because it is necessary to guarantee long-time jobs (sensor monitoring), including when users are offline, save the results file, and signal the jobs that are complete. It was decided to adopt OGC **Sensor Observation Service** (**SOS**) standard, to retrieve the results from a database and forward them to the user when requested. Therefore the database is populated by the CJI, according to the users' requests, and extracts those results using a SOS web service.

It is also the CJI that receives the user requests and processes them into multiple single sensor readings, marking those readings with a tag that will identify the user and job to which they belong.

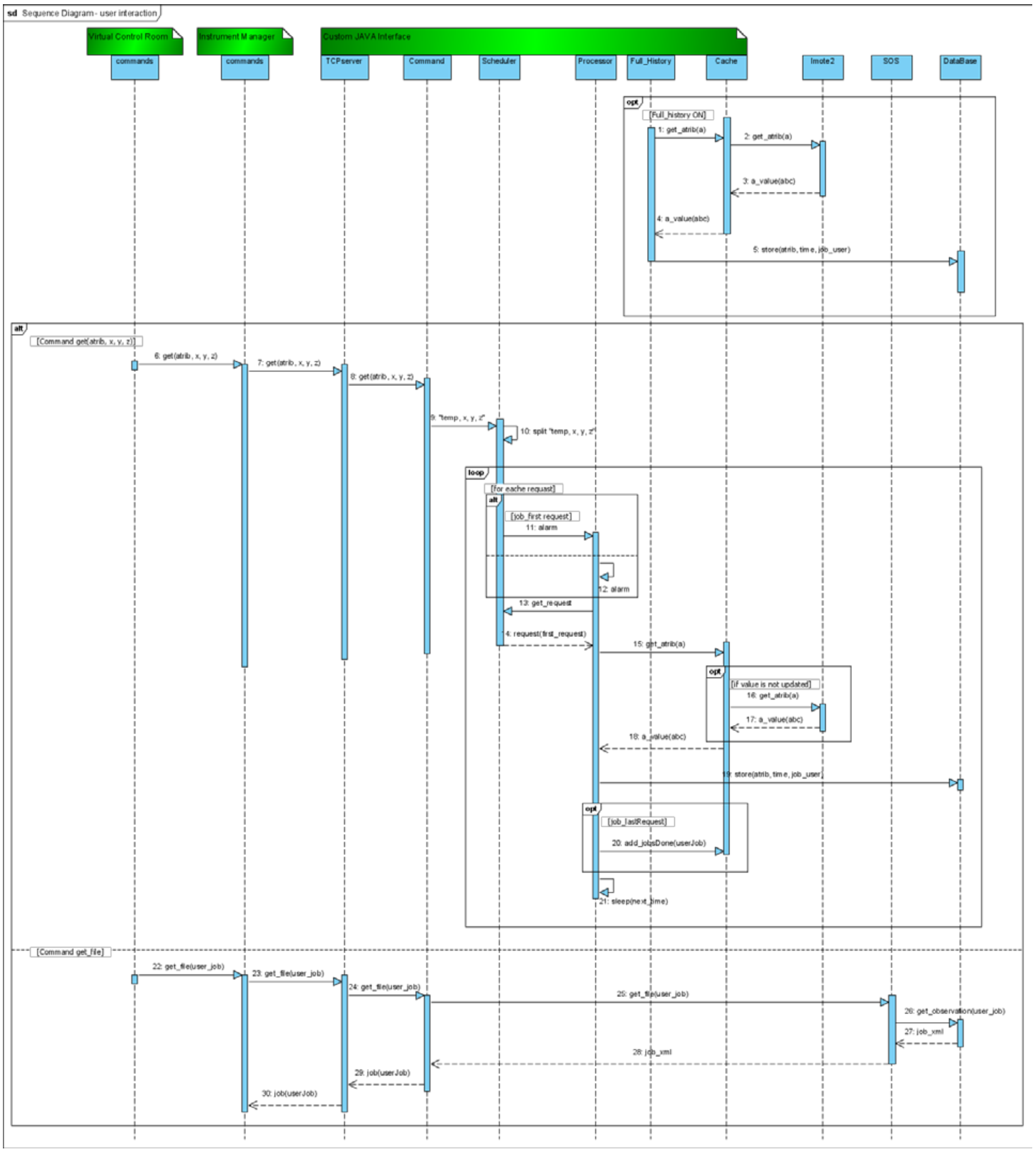

Figure 9 – User interaction mode timing (*get_atrib* and *get file* commands).

As mentioned earlier, our sensors are Imote2 boards with a temperature sensor, a light sensor, an accelerometer and a humidity sensor. These boards are programmed with custom nesC code running on TinyOS 1.x, so they can answer our requests.

The Virtual Control Room (VCR) provides the user with several commands to interact with the sensors, and show the state of some defined attributes and parameters. This state is updated every 5 seconds by the VCR from the Instrument Manager.

We currently have the following **attributes**:

- Temperature;
- Humidity;
- Light;
- Accelerometer;
- Battery level;
- Jobs done;
- File URL.

And **parameters**:

- ID;
- Latitude;
- Longitude.

Finally, the **commands** available to the users are:

- Get(<temperature | humidity | light | accelerometer>, each *time*, during *time2*, *job-name*);
- Get job(*job-name*).

All the attributes have a lifetime, depending on the related phenomenon: e.g. accelerometer-1 second, temperature-15 minutes, etc.
When a job is complete, the flag array "jobsDone" is updated with the job identification, so the user who ordered it knows his file is ready. Then the user may "call" the results file and get a URL to the file containing the observations, which he could use or save in some place.

The **Instrument Manager** (**IM**) is an object that virtualizes an instrument (in this case an Imote2 node), and performs the required translation between the instrument and the user-VCR, in order to operate and monitor the sensors.
The IM has a cache, where the values for the VCR attributes and parameters are kept, and this cache is automatically updated every 20 milliseconds with values from the instrument.
In this prototype, and due to the peculiarity and characteristics of the instrument (Imote2 sensor node) and its normal way of use, an intermediate element is used before the instrument itself. The CJI is used to implement another cache to answer to the IM, therefore avoiding unnecessary and possibly problematic interactions with the sensors, due to energy and bandwidth management issues.

One of the issues resulting from the use of VCR-IE is that, if any user forgets or intentionally closes the VCR without logging out, the IE/IM still works *ad eternum*. This issue has the following major consequences:

- Energy management crashes due to the non-ending polling to the CJI and sensors;
- When this user reconnects to the VCR, he has all the data (relative to all attributes), from that whole period available in the VCR, ready for use, without the need for programming periodic observations.

The CJI has several components in order to manage the system needs:

- A TCP server to manage the connection to the IM;
- A command block to parse the commands and redirect them to the adequate recipient;

- A scheduler to split a periodic job into several unique requests and insert them in a job schedule table;
- A processor to process the requests from the table and save the results in a database;
- A cache to keep the sensor values and update them when their lifetime ends;
- An historical module to keep a complete log of all the attributes in the database.

The CJI has two logging modes: “history on” and “history off”. The “history off” mode only saves the results from the user requests on the database, while the “history on” mode saves all the readings made by the sensors, according to the lifetime of each attribute.

The attribute lifetime, the logging mode and the battery management are configurable and controlled by the “**I**nstrument **A**ccess **S**ervices **H**ost” administrator using a XML configuration file (modifications in this XML file will need a system restart to become effective).

4 Case Study: Implementation of a WebService to Access an Imote2 Sensor Network According to the OGC Standards

To overcome some of the limitations of GRICC/DORII instrument access, a second case study was carried out: the implementation of a WebService, to access Imote2 sensors, based on the OGC standards.

4.1 Summary of the OGC and SensorWeb Initiative

OGC [15] is an international industry consortium of more than 300 companies, government agencies, research organizations, and universities, participating in a consensus process to develop publicly available interface specifications.
In the *OGC Sensor Web Enablement* (SWE) activity [16], members of the OGC are defining, testing, and documenting a consistent framework of open standards for exploiting Web-connected sensors and sensor systems of any type. Sensor Web Enablement presents many opportunities for adding a real-time sensor dimension to the Internet and the Web. This has extraordinary significance for science, environmental monitoring, transportation management, public safety, facility security, disaster management, industrial controls, facilities management and many other domains of activity.

The OGC voluntary consensus standards setting process, coupled with strong international industry and government support, in domains that depend on sensors is expected to result in SWE specifications that will become established in all application areas where such standards are of use.

4.2 Related Work

The work of some institutes has been studied, such as NICTA/ Melbourne University and 52ºNorth, which are two solid examples in terms of OGC standards implementation.

NICTA/Melbourne University

At NICTA (National ICT Australia Ltd) University of Melbourne, there is an effort, called the **OSWA** (**O**pen **S**ensor **W**eb **A**rchitecture), [17] to utilize the combination of SOA, Grid computing and sensor networks by means of OGC standards. It aims at providing the middleware support and programming environment for creating, accessing and utilizing sensor services through the Web.
This integration has brought several benefits to the community. First of all, the heavy load of information processing can be moved from sensor networks to the backend-distributed systems such as Grids. Individual sensor networks can be linked together as services, which can be registered, discovered and accessed by different clients using a uniform protocol. Moreover, Grid-based sensor applications can provide advanced services for smart-sensing by developing scenario-specific operators at runtime. Figure 10 shows an overview of the OSWA architecture [17].

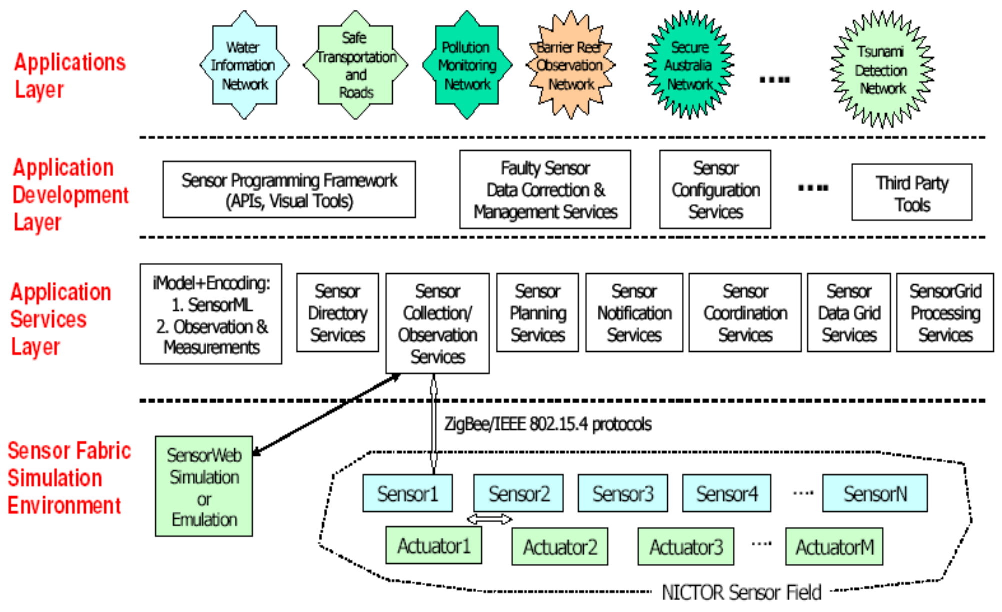
Applications Layer
Water Information Network
Safe Transportation and Roads
Pollution Monitoring Network
Barrier Reef Observation Network
Secure Australia Network
Tsunami Detection Network
Application Development Layer
Sensor Programming Framework (APIs, Visual Tools)
Faulty Sensor Data Correction & Management Services
Sensor Configuration Services
Third Party Tools
Application Services Layer
iModel+Encoding: 1. SensorML 2. Observation & Measurements
Sensor Directory Services
Sensor Collection/ Observation Services
Sensor Planning Services
Sensor Notification Services
Sensor Coordination Services
Sensor Data Grid Services
SensorGrid Processing Services
ZigBee/IEEE 802.15.4 protocols
Sensor Fabric Simulation Environment
SensorWeb Simulation or Emulation
Sensor1
Sensor2
Sensor3
Sensor4
SensorN
Actuator1
Actuator2
Actuator3
ActuatorM
NICTOR Sensor Field

Figure 10 – OSWA layers architecture.

52°North

52°North Initiative for Geospatial Open Source Software GmbH is an international research and development company whose mission is to promote the conception, development and application of free open source geo-software for research, education, training and practical use [18].
52°North's Sensor Web community focuses on the development of a broad range of services and encoding implementations related to SWE. Their vision is to enable real time integration of heterogeneous sensor Webs into the information infrastructure and their mission is to help build the Sensor Web as a global infrastructure for observation data while acknowledging the "locality" of some sensor networks.
The community is intensively involved in the Open Geospatial Consortium's standardization process, which will ensure a sustainable development. OGC's Sensor Web Enablement

initiative aims at standardizing the entire sensor web process: description of sensors and the setup of sensor and measurement registries, discovery and sensor tasking mechanisms, and access of data observed by sensors. 52°North accelerates the entire process of setting up interoperable sensor Webs by bundling developer capacities.

4.3 Analysis of the Basic Standards (O&M, SensorML/GML) for Modeling and Codification of the Involved Information in the Access to Sensors

SensorML

SensorML [19] provides a functional model of the sensor system, rather than a detailed description of its hardware. SensorML treats sensor systems and a system's components (e.g. sensors, actuators, platforms, etc.) as processes. It provides additional metadata that are useful for enabling discovery, for identifying system constraints (e.g. security or legal use constraints), for providing contacts and references, and for describing taskable properties, interfaces, and physical properties.
Figure 11 presents the conceptual model of the data type that serves as the basis for specifying all inputs, outputs and parameters within a process: the simple data type.

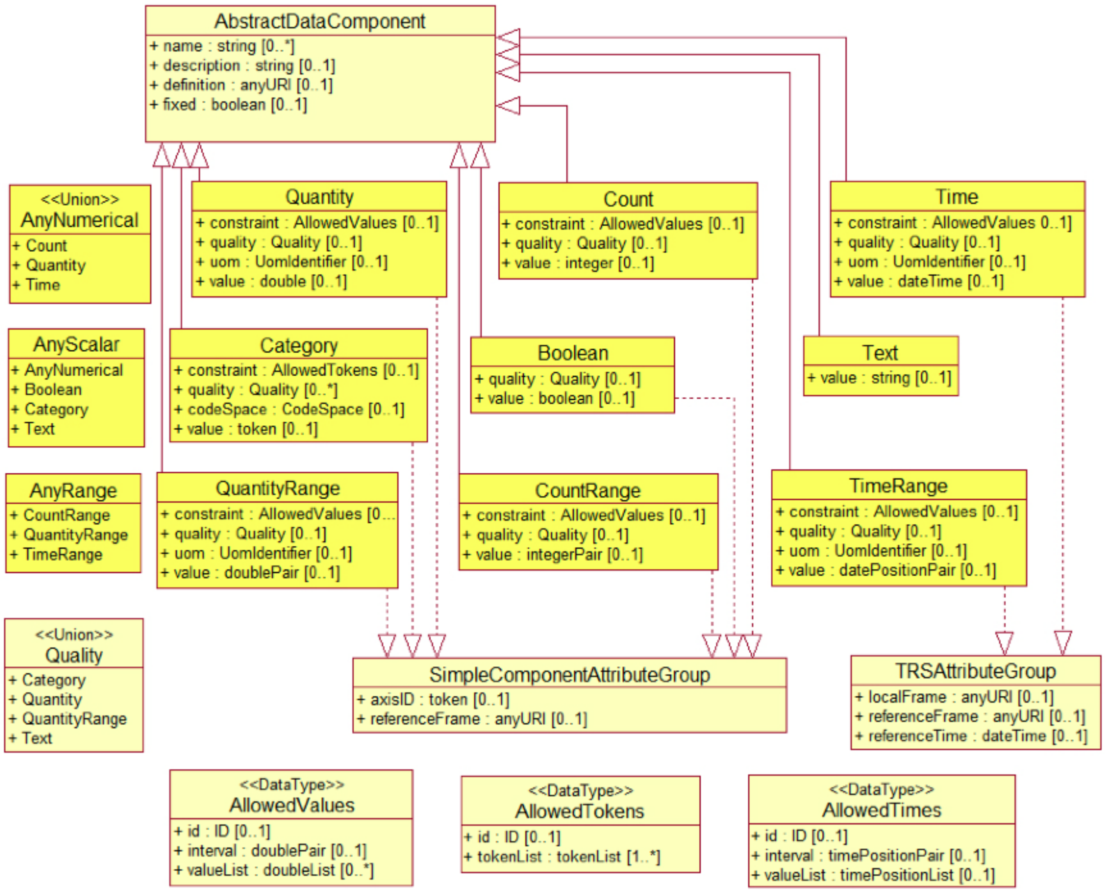
AbstractDataComponent
+ name : string [0..*]
+ description : string [0..1]
+ definition : anyURI [0..1]
+ fixed : boolean [0..1]
<<Union>>
AnyNumerical
+ Count
+ Quantity
+ Time
Quantity
+ constraint : AllowedValues [0..1]
+ quality : Quality [0..1]
+ uom : UomIdentifier [0..1]
+ value : double [0..1]
Count
+ constraint : AllowedValues [0..1]
+ quality : Quality [0..1]
+ value : integer [0..1]
Time
+ constraint : AllowedValues 0..1]
+ quality : Quality [0..1]
+ uom : UomIdentifier [0..1]
+ value : dateTime [0..1]
AnyScalar
+ AnyNumerical
+ Boolean
+ Category
+ Text
Category
+ constraint : AllowedTokens [0..1]
+ quality : Quality [0..*]
+ codeSpace : CodeSpace [0..1]
+ value : token [0..1]
Boolean
+ quality : Quality [0..1]
+ value : boolean [0..1]
Text
+ value : string [0..1]
AnyRange
+ CountRange
+ QuantityRange
+ TimeRange
QuantityRange
+ constraint : AllowedValues [0...
+ quality : Quality [0..1]
+ uom : UomIdentifier [0..1]
+ value : doublePair [0..1]
CountRange
+ constraint : AllowedValues [0..1]
+ quality : Quality [0..1]
+ value : integerPair [0..1]
TimeRange
+ constraint : AllowedValues [0..1]
+ quality : Quality [0..1]
+ uom : UomIdentifier [0..1]
+ value : datePositionPair [0..1]
<<Union>>
Quality
+ Category
+ Quantity
+ QuantityRange
+ Text
SimpleComponentAttributeGroup
+ axisID : token [0..1]
+ referenceFrame : anyURI [0..1]
TRSAttributeGroup
+ localFrame : anyURI [0..1]
+ referenceFrame : anyURI [0..1]
+ referenceTime : dateTime [0..1]
<<DataType>>
AllowedValues
+ id : ID [0..1]
+ interval : doublePair [0..1]
+ valueList : doubleList [0..*]
<<DataType>>
AllowedTokens
+ id : ID [0..1]
+ tokenList : tokenList [1..*]
<<DataType>>
AllowedTimes
+ id : ID [0..1]
+ interval : timePositionPair [0..1]
+ valueList : timePositionList [0..1]

Figure 11 - The SensorML simple data type conceptual model.

Observation and Measurements (O&M)

Observation & Measurements [20] defines a framework, a conceptual model and an encoding formalized as an Application Scheme, but is applicable across a wide variety of applications domain. O&M binds a result to a feature of interest upon which the observation was made using a procedure to determine the value of the result. The result is an estimate of the value of a property of the feature of interest. The basic Observation Model Type describes four key properties of an observation:

- ***featureOfInterest***: A feature of any type (ISO 19109, ISO 19101), which is a representation of the observation target, being the real-world *object* regarding which the observation is made.
- ***observedProperty***: It identifies or describes the phenomenon for which the observation result provides an estimate of its value. It must be a *property* (phenomenon) associated with the type of feature of interest.
- ***procedure:*** The description of a process used to generate the result. It must be suitable for the observed property. It does not distinguish between sensor observations, estimations made by an observer, or algorithms, simulations, computations and complex processing chains.
- ***result:*** It contains the value generated by the procedure. The type of the observation result must be consistent with the observed property, and the scale or scope for the value must be consistent with the quantity or category type. *Result* is an estimate of the value of some property of the feature of interest.

We can see the O&M model in figure 12.

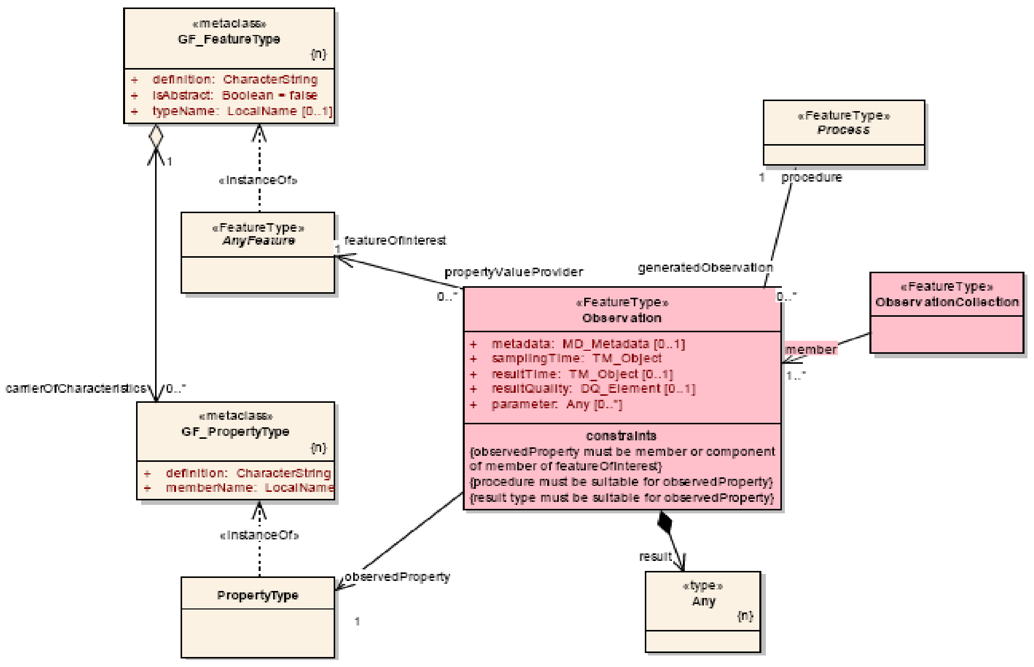
«metaclass»
GF_FeatureType
{n}
+ definition: CharacterString
+ isAbstract: Boolean = false
+ typeName: LocalName [0..1]
1
«instanceOf»
«FeatureType»
AnyFeature
featureOfInterest
1
«FeatureType»
Process
1 procedure
propertyValueProvider
0..*
generatedObservation
0..*
«FeatureType»
Observation
+ metadata: MD_Metadata [0..1]
+ samplingTime: TM_Object
+ resultTime: TM_Object [0..1]
+ resultQuality: DQ_Element [0..1]
+ parameter: Any [0..*]
constraints
{observedProperty must be member or component of member of featureOfInterest}
{procedure must be suitable for observedProperty}
{result type must be suitable for observedProperty}
«FeatureType»
ObservationCollection
member
1..*
carrierOfCharacteristics
0..*
«metaclass»
GF_PropertyType
{n}
+ definition: CharacterString
+ memberName: LocalName
«instanceOf»
PropertyType
observedProperty
1
result
«type»
Any
{n}

Figure 12 – The O&M model.

The O&M model contains several additions to make the Observation concept more compatible with the sensor data, with new complex types such as *ObservationArray* and *ObservationCollection*, for defining time-series data fetched, thus the data retrieved from the sensors would easily be described.

4.4 Analysis of the Standards that Define Essential Services to Access the Sensors Data (SOS and SPS)

In this section the standard services that are essential to access the sensors data will be summarized.

Sensor Observation Service (SOS)

The SOS [21] service provides an interface to make sensors and sensor data archives accessible via an interoperable Web based interface. See figure 13 for the time sequence of the allowed SOS operations.

- **GetCapabilities**: for requesting a self-description of the service (request the capabilities file of the service);
- **DescribeSensor**: for requesting information about the sensor itself, encoded in a SensorML instance document;
- **GetObservation**: for requesting the pure sensor data, encoded in O&M.

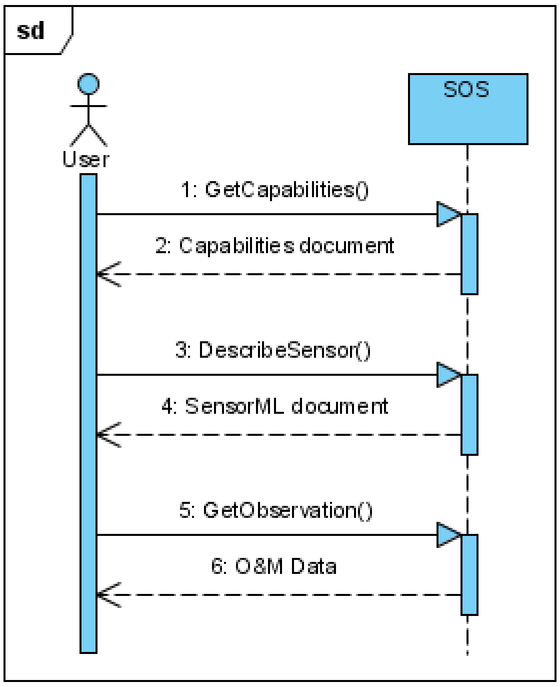

Figure 13 – The sequence diagram for the SOS operations.

Sensor Planning Service (SPS)

The SPS [22] is an interface to task an asset or asset system. It is not an interface to access the observational data produced by it. There are some mandatory operations of this service:

- **GetCapabilities**: to retrieve metadata about the capabilities of an SPS.
- **DescribeTasking:** to obtain the information that is required to prepare a tasking request.
- **GetFeasibility**: to obtain the feasibility details about a tasking request. The task might not be fulfilled because the sensor maybe in use.
- **Submit**: to submit a tasking request.
- **DescribeResultAccess**: to retrieve information about where observed data can be accessed from. This may be from an SOS, for example.

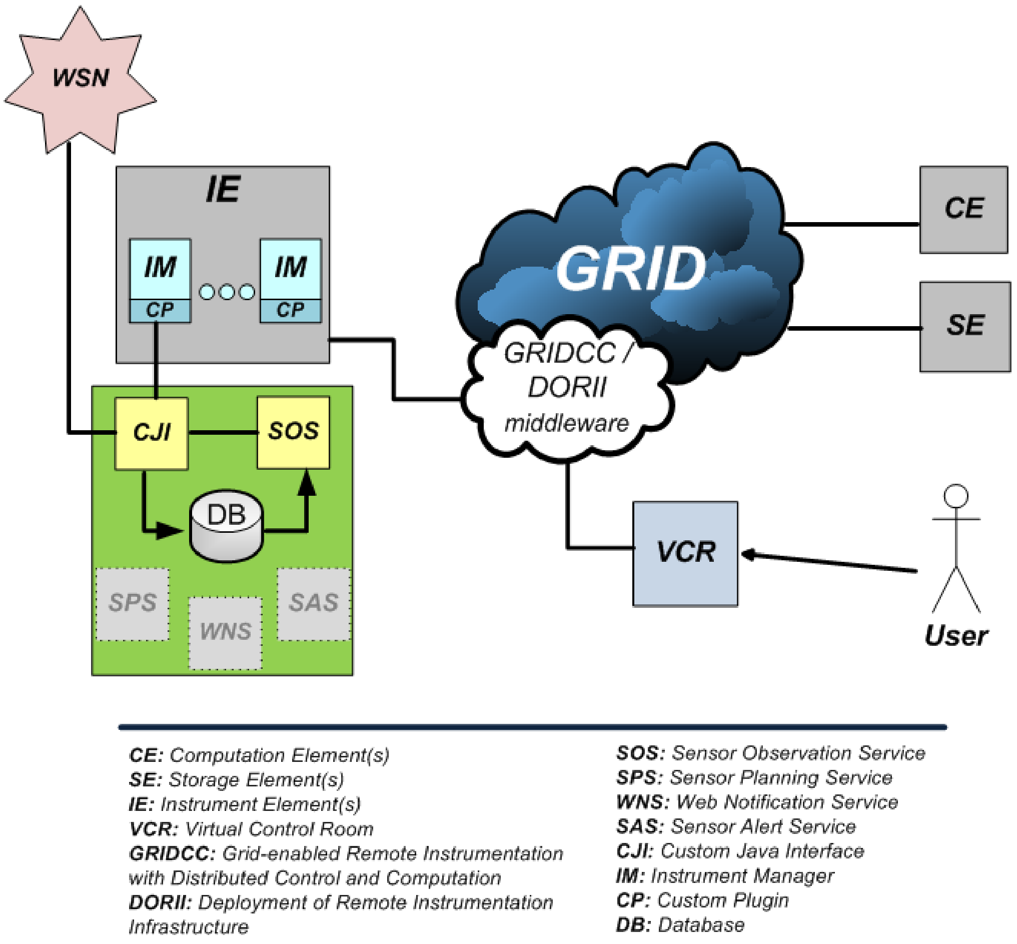
WSN
IE
IM
CP
IM
CP
CJI
SOS
DB
SPS
WNS
SAS
GRID
GRIDCC / DORII middleware
CE
SE
VCR
User
CE: Computation Element(s)
SE: Storage Element(s)
IE: Instrument Element(s)
VCR: Virtual Control Room
GRIDCC: Grid-enabled Remote Instrumentation with Distributed Control and Computation
DORII: Deployment of Remote Instrumentation Infrastructure
SOS: Sensor Observation Service
SPS: Sensor Planning Service
WNS: Web Notification Service
SAS: Sensor Alert Service
CJI: Custom Java Interface
IM: Instrument Manager
CP: Custom Plugin
DB: Database

Figure 14 – A simplified view of the architecture including the access to sensor data through the SOS OGC standard.

4.5 The Proposed CYCLOPS Architecture

The proposed architecture is based on reading, controlling and analyzing data (observations of a natural phenomenon) from a sensor network through an environment Grid, to support civil protection applications.

As in the previous case study, the GRIDCC middleware was also used for accessing the Grid in conjunction with the OGC standards sensor observation services. In this architecture, the access to a controllable database is carried out by the previously presented CJI (Custom Java Interface). Figure 14 presents the scheme of the proposed architecture.

All the services of the OGC, as well as the control service CJI and the database, are available through the IM of the GRIDCC whose access is by means of the VCR.

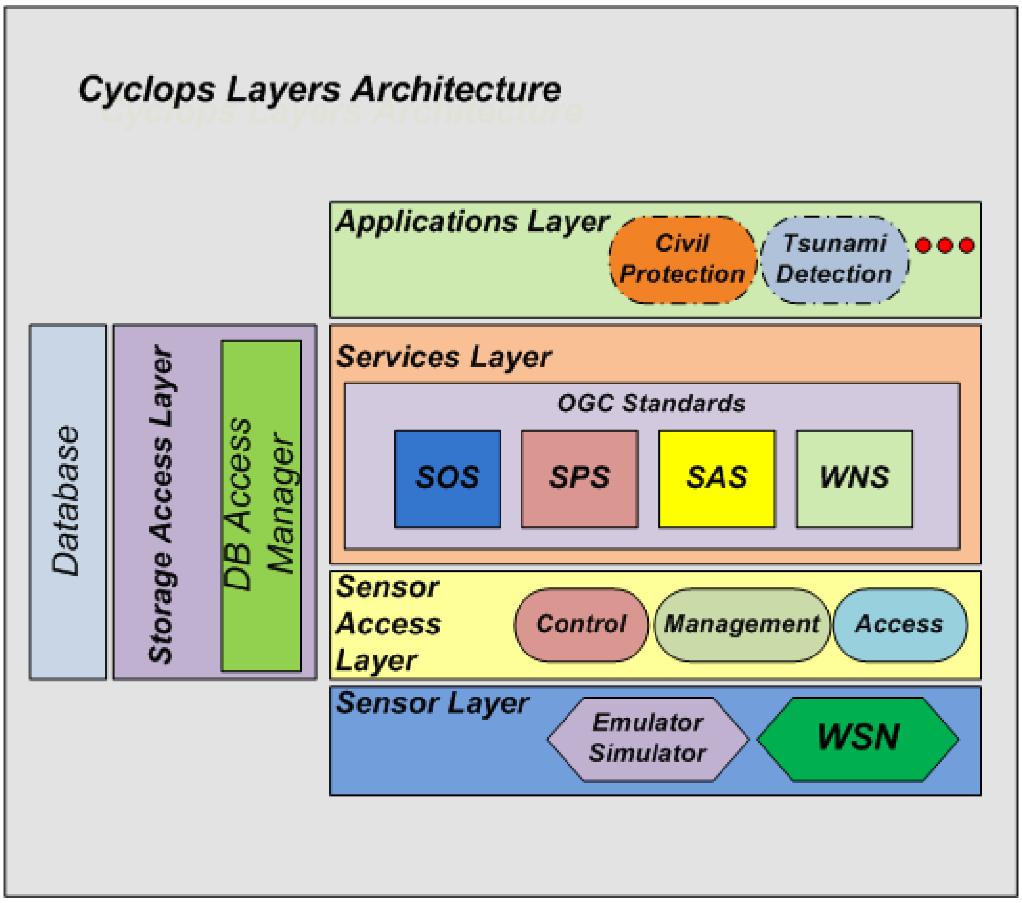

Figure 15 – Cyclops layers architecture following the OGC approach.

An OGC proposal, not fully implemented, is illustrated by the layered architecture included in figure 15. Service and sensor access layers can access the database. In this case, the CJI functionalities would be spread between the service layer (observations scheduling), the sensor access layer (instrument control, access and management) and the storage access layer (observations saving).

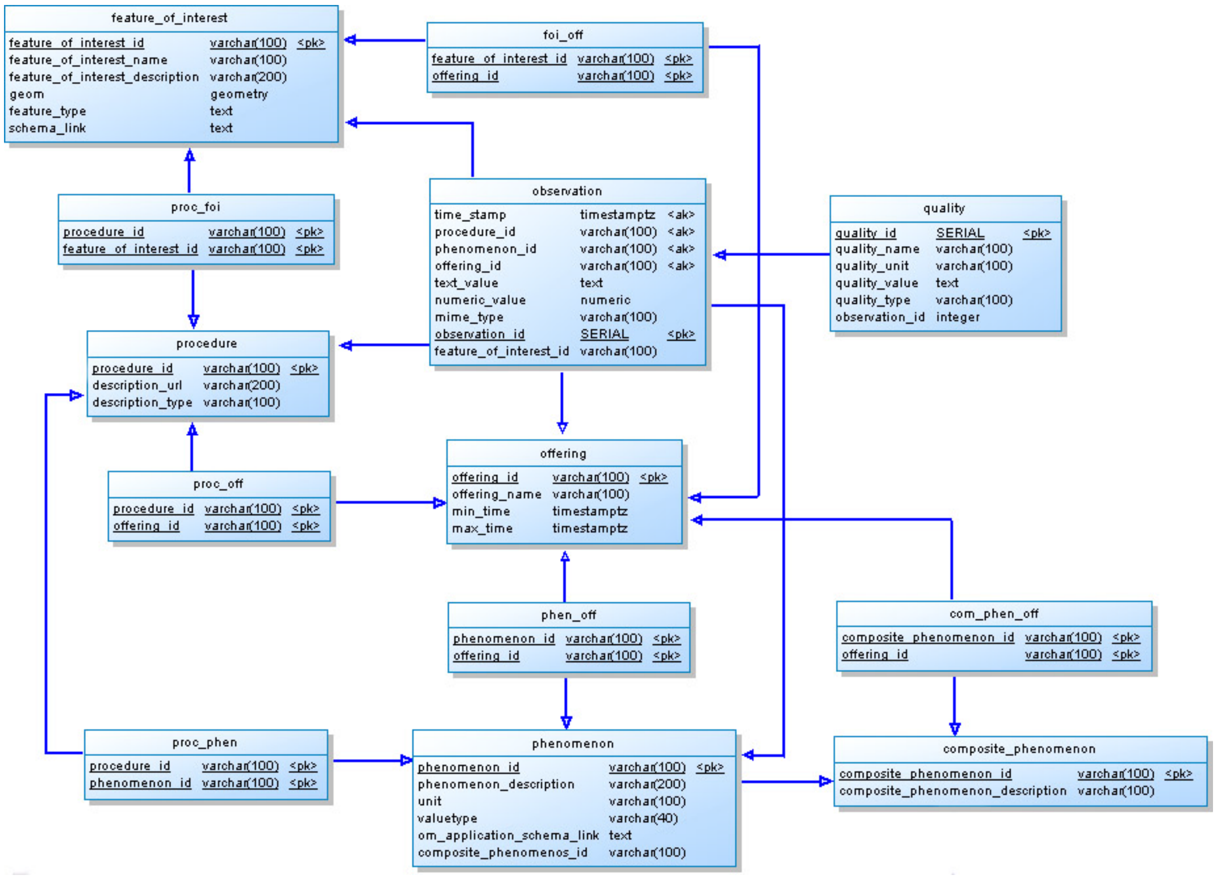
feature_of_interest
feature_of_interest_id varchar(100) <pk>
feature_of_interest_name varchar(100)
feature_of_interest_description varchar(200)
geom geometry
feature_type text
schema_link text
foi_off
feature_of_interest_id varchar(100) <pk>
offering_id varchar(100) <pk>
proc_foi
procedure_id varchar(100) <pk>
feature_of_interest_id varchar(100) <pk>
observation
time_stamp timestamptz <ak>
procedure_id varchar(100) <ak>
phenomenon_id varchar(100) <ak>
offering_id varchar(100) <ak>
text_value text
numeric_value numeric
mime_type varchar(100)
observation_id SERIAL <pk>
feature_of_interest_id varchar(100)
quality
quality_id SERIAL <pk>
quality_name varchar(100)
quality_unit varchar(100)
quality_value text
quality_type varchar(100)
observation_id integer
procedure
procedure_id varchar(100) <pk>
description_url varchar(200)
description_type varchar(100)
offering
offering_id varchar(100) <pk>
offering_name varchar(100)
min_time timestamptz
max_time timestamptz
proc_off
procedure_id varchar(100) <pk>
offering_id varchar(100) <pk>
phen_off
phenomenon_id varchar(100) <pk>
offering_id varchar(100) <pk>
com_phen_off
composite_phenomenon_id varchar(100) <pk>
offering_id varchar(100) <pk>
proc_phen
procedure_id varchar(100) <pk>
phenomenon_id varchar(100) <pk>
phenomenon
phenomenon_id varchar(100) <pk>
phenomenon_description varchar(200)
unit varchar(100)
valuetype varchar(40)
om_application_schema_link text
composite_phenomenos_id varchar(100)
composite_phenomenon
composite_phenomenon_id varchar(100) <pk>
composite_phenomenon_description varchar(100)

Figure 16 – The Cyclops database.

The Database used in this proposal is based on the database proposal presented by the 52ºNorth [18] initiative which is included in figure 16. Figure 16 shows the data tables of the SOS database. The SOS will use the tables above to answer incoming requests. However, the SOS is not responsible for updating the values in these tables. This task has to be fulfilled externally, using the CJI.

The core tables and definitions are described next:

feature_of_interest – The *feature_of_interest* table stores data about the feature of interest. The *geom* column holds the geometry of the *feature_of_interest*. Feature(s) represent(s) the identifiable object(s) on which the sensor systems are making observations. In the case of an in-situ sensor this may be a station to which the sensor is attached, representing the environment directly surrounding the sensor. For remote sensors, this may be the area or volume that is being sensed, which is not co-located with the sensor. For example, a sensor, a place, a station weather, etc.

observation – The observation table aggregates the data of an observation event such as time, procedure (sensor or group of sensors), the feature of interest and the observation value, which is stored in a separate table. An observation is an action with a result which has a value describing some phenomenon. An observation uses a procedure to determine the value of the result, which may involve a sensor or observer, analytical procedure, simulation or other numerical process.

quality – The quality table stores quality attributes for an observation. Qualities are optional and do not have to be set.

procedure – The procedure table stores data about the procedure. Only the *procedure_id*, which should be the **URN** (**U**niform **R**esource **N**ame) of the procedure as specified by the OGC, must be contained. Procedure is the reference to one or more procedures, including sensor systems, instruments, simulators, etc, that supply observations in this offering. The *DescribeSensor* operation can be called to provide a *SensorML* or *TransducerML* description for each system or procedure.

offering – The offering table stores each offering of this SOS. This table is only used when the SOS is initialized to read in the offerings of this SOS and the phenomena which are related to each offering. An SOS organizes collections of related sensor system observations into Observation offerings. For example, the temperature in a Weather Station placed in a City.

phenomenon – The phenomenon table represents phenomena (temperature, humidity, etc). Only the *phenomenon_id* and *value_type* are required. The *phenomenon_id* should contain the URN of the phenomenon as specified by the OGC. Each phenomenon has a unit of measure. For example, the temperature unit is Celsius.

composite_phenomenon – The *composite_phenomenon* stores composite phenomena.

The other tables represent the link with a many-to-many relationship between two core tables.

4.6 Implementation of the SOS Service

It was decided to implement a basic SOS. The SOS receives XML requests and returns XML documents as a response. The *GetObservation* operation accesses the database and returns the observation requested. A database example is shown next.

Database Example:

- 2 Features of Interest (Wheatear Stations)
  → *Geres North – id_N*
  → *Geres South – id_S*

- 5 Simple Phenomena (Observed Properties)
  → *Air Temperature*
  → *Relative Humidity*
  → *Radiation (luminous intensity)*
  → *Wind Direction*
  → *Wind Speed*

- 1 Composite Phenomenon (Observed Property)
  → *Wheater*
  1. Air temperature
  2. Relative Humidity
  3. Radiation

- 10 Simple Observation Offerings (*2 features * 5 phenomena)*

  1. *GERES_NORTH_TEMPERATURE*
  2. *GERES_NORTH_HUMIDITY*
  3. *GERES_NORTH_RADIATION*
  4. *GERES_NORTH_WIND_DIRECTION*
  5. *GERES_NORTH_WIND_SPEED*
  6. *GERES_SOUTH_TEMPERATURE*
  7. *GERES_SOUTH_HUMIDITY*
  8. *GERES_SOUTH_RADIATION*
  9. *GERES_SOUTH_WIND_DIRECTION*
  10. *GERES_SOUTH_WIND_SPEED*

- 2 Composed Observation Offerings *(2 features * 1 composed phenomenon)*
  1. *GERES_NORTH_WHEATER*
  2. *GERES_NORTH_WHEATER*

-- composite phenomenon
INSERT INTO composite_phenomenon VALUES ('compPhen1', 'Weather on GERES');

-- sample phenomenon
INSERT INTO phenomenon (phenomenon_id, phenomenon_description, unit, valuetype, composite_phenomenon_id)
VALUES ('urn:x-ogc:def:phenomenon:OGC:AirTemperature', 'AirTemperature', 'urn:x-ogc:def:unit:degC','numericType','compPhen1');
INSERT INTO phenomenon (phenomenon_id, phenomenon_description, unit, valuetype, composite_phenomenon_id)
VALUES ('urn:x-ogc:def:phenomenon:OGC:RelativeHumidity', 'RelativeHumidity', 'urn:x-ogc:def:unit:percent','numericType','compPhen1');
INSERT INTO phenomenon (phenomenon_id, phenomenon_description, unit, valuetype, composite_phenomenon_id)
VALUES ('urn:x-ogc:def:phenomenon:OGC:Radiation', 'Radiation', 'urn:x-ogc:def:unit:cd','numericType','compPhen1');
INSERT INTO phenomenon (phenomenon_id, phenomenon_description, unit, valuetype)
VALUES ('urn:x-ogc:def:phenomenon:OGC:WindDirection', 'WindDirection', 'urn:x-ogc:def:unit:degree','numericType');
INSERT INTO phenomenon (phenomenon_id, phenomenon_description, unit, valuetype)
VALUES ('urn:x-ogc:def:phenomenon:OGC:WindSpeed', 'WindSpeed', 'urn:x-ogc:def:unit:meterPerSecond','numericType');

-- sample offering
INSERT INTO offering VALUES ('GERES_NORTH_TEMPERATURE','The Temperature in North of GERES',null,null);
INSERT INTO offering VALUES ('GERES_NORTH_HUMIDITY','The Humidity in North of GERES',null,null);
INSERT INTO offering VALUES ('GERES_NORTH_RADIATION','The Radiation in North of GERES',null,null);
INSERT INTO offering VALUES ('GERES_NORTH_WIND_DIRECTION','The Wind Direction in North of GERES',null,null);
INSERT INTO offering VALUES ('GERES_NORTH_WIND_SPEED','The Wind Speed in North of GERES',null,null);
INSERT INTO offering VALUES ('GERES_SOUTH_TEMPERATURE','The Temperature in South of GERES',null,null);
INSERT INTO offering VALUES ('GERES_SOUTH_HUMIDITY','The Humidity in South of GERES',null,null);
INSERT INTO offering VALUES ('GERES_SOUTH_RADIATION','The Radiation in South of GERES',null,null);
INSERT INTO offering VALUES ('GERES_SOUTH_WIND_DIRECTION','The Wind Direction in South of GERES',null,null);
INSERT INTO offering VALUES ('GERES_SOUTH_WIND_SPEED','The Wind Speed in South of GERES',null,null);
INSERT INTO offering VALUES ('GERES_NORTH_WHEATER','The Weather in North of GERES',null, null);
INSERT INTO offering VALUES ('GERES_SOUTH_WHEATER','The Weather in South of GERES',null, null);

-- sample featureofinterest
INSERT INTO feature_of_interest (feature_of_interest_id, feature_of_interest_name, feature_of_interest_description, geom, feature_type, schema_link)
VALUES ('id_N', 'GERES_NORTH', 'North', GeometryFromText('POINT(7.727958 51.883906)', 4326),'sa:SamplingPoint', 'http://xyz.org/reference-url2.html');
INSERT INTO feature_of_interest (feature_of_interest_id, feature_of_interest_name, feature_of_interest_description, geom, feature_type, schema_link)
VALUES ('id_S', 'GERES_SOUTH', 'South', GeometryFromText('POINT(8.76667 51.7167)', 4326),'sa:SamplingPoint', 'http://xyz.org/reference-url2.html');

-- sample procedure
INSERT INTO procedure VALUES ('urn:ogc:object:feature:Sensor:UM:um-sensor-north', 'standard/um-sensor-north.xml', 'text/xml;subtype="SensorML"');
INSERT INTO procedure VALUES ('urn:ogc:object:feature:Sensor:UM:um-sensor-south', 'standard/um-sensor-south.xml', 'text/xml;subtype="SensorML"');

SOS Operations

Below we can see SOS commands request examples. Due to the length of the responses, these are included in appendix II.

GetCapabilities request

```
<?xml version="1.0" encoding="UTF-8"?>
<GetCapabilities xmlns="http://www.opengis.net/sos/1.0"
xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sos/1.0
http://schemas.opengis.net/sos/1.0.0/sosGetCapabilities.xsd" service="SOS"
updateSequence="">
<ows:AcceptVersions>
        <ows:Version>1.0.0</ows:Version>
</ows:AcceptVersions>
<ows:Sections>
        <ows:Section>OperationsMetadata</ows:Section>
        <ows:Section>ServiceIdentification</ows:Section>
        <ows:Section>Contents</ows:Section>
</ows:Sections>
</GetCapabilities>
```

DescribeSensor request

```
<?xml version="1.0" encoding="UTF-8"?>
<DescribeSensor xmlns="http://www.opengis.net/sos/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sos/1.0
http://schemas.opengis.net/sos/1.0.0/sosDescribeSensor.xsd"  service="SOS"
outputFormat="text/xml;subtype=&quot;SensorML/1.0.0&quot;"  version="1.0.0">
  <procedure>urn:ogc:object:feature:Sensor:UM:um-sensor-north</procedure>
</DescribeSensor>
```

GetObservation request

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObservation xmlns="http://www.opengis.net/sos/1.0"
xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:gml="http://www.opengis.net/gml"
xmlns:ogc="http://www.opengis.net/ogc" xmlns:om="http://www.opengis.net/om/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sos/1.0
http://schemas.opengis.net/sos/1.0.0/sosGetObservation.xsd"
service="SOS" version="1.0.0" >
<offering>GERES_NORTH_TEMPERATURE</offering>
<observedProperty>urn:x-ogc:def:phenomenon:OGC:AirTemperature</observedProperty>
<responseFormat>text/xml;subtype=&quot;om/1.0.0&quot;</responseFormat>
</GetObservation>
```

5 Conclusions

A functional prototype to provide Grid civil protection applications with data from a remote wireless sensor network was developed. A first version of the prototype was based on the utilization of the GRIDCC/DORRI middleware. This prototype was later improved with observations handling and services which are in conformity with the OGC SWE standards, O&M/SensorML and SOS respectively. SOS was implemented successfully, working in Standalone mode, receiving XML requests and returning XML documents with the response.

From our (CYCLOPS) point of view, the main weaknesses of the present DORII middleware evolution are:

- The necessity of having the VCR online in order to receive alarms or to carry out planned observations;
- The network instrument issue – the constraints in the network virtualization;
- Control needs to be shared simultaneously by several different users while maintaining the integrity of their individual jobs.

These problems will be overcome if an empowered interface is used in the instrument side, but this will result in custom and non-normalized libraries, and maybe some problems regarding GRID protocols. In order to provide a feasible CYCLOPS tool, these points should be corrected in DORII, as well as a few others of minor functionality already discussed when a member from our team was with the DORII team at *Electtra*. In the present stage of development, the IE concept has shown itself to be a little too restrictive to be used by CP people, however, it seems that there are good chances of it evolving in the right direction in the near future.

The Web-based access to sensors seems to be an obvious valuable improvement to the local and traditional access to heterogeneous sensors. Further studies are needed to evaluate if Grid enabled access to sensors is adequate for Civil Protection purposes. It is necessary to identify and balance the real necessities of CP people, the benefits brought by this technology, the training overhead needed to use it, and its fidelity in critical situations usage.

In the future, a QoS could be defined to guarantee that Grid-enabled applications and sensor accessing fulfill Civil Protection requirements. QoS can be used to guarantee that applications perform in real-time, or near real-time. Another issue to address in the future is power consumption control in sensors. In WSNs, like those based in Imote2s, power is a critical resource that must be saved as much as possible. This can be crucial to keeping sensors running when an intensive access, such as a full time access, is being performed by CP applications in a emergency situation.

6 References

[1] The Cyclops Project web site. http://www.cyclops-project.eu

[2] The DORII project: Deployment of Remote Instrumentation Infrastructure. http://www.dorii.eu

[3] Xinjie Chang. Network Simulations with OPNET.

[4] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. *Proceedings of SenSys'03, First ACM Conference on Embedded Networked Sensor Systems, New York*, ACM Press, 2003.

[5] C. Mallanda, A. Suri, V. Kunchakarra, S.S. Iyengar, R. Kannan, and A. Durresi. Simulating Wireless Sensor Networks with OMNeT++.

[6] Sung Park, Andreas Savvides, and Mani B. Srivastava. Sensorsim: a simulation framework for sensor networks, in MSWIM'00: Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems, (New York, NY, USA), pp. 104–111, ACM Press, 2000.

[7] The Network Simulator – ns-2. http://www.isi.edu/nsnam/ns.

[8] Gilbert Chen, Joel Branch, Michael Pflug, Lijuan Zhu, and Boleslaw Szymanski. SENSE: A Sensor Network Simulator. *Advances in Pervasive Computing and Networking*, 2004.

[9] Jonathan Pollet, et al. ATEMU: A Fine-Grained Sensor Network Simulator. *Proceedings of SECON'04, First IEEE Communications SocietyConference on Sensor and Ad Hoc Communications and Networks*, 2004.

[10] Sameer Sundresh, Wooyoung Kim, and Gul Agha. SENS: A Sensor, Environment, and Network Simulator. *Proceedings of 37th Annual Simulation Symposium*, 2004.

[11] Ben L. Titzer, Daniel K. Lee, Jens Palsberg. Avrora: Scalable Sensor Network Simulation With Precise Timing. *Proceedings of IPSN'05,Fourth International Conference on Information Processing in Sensor Networks*, 2005.

[12] TinyOS Tutorial: http://www.tinyos.net/tinyos-1.x/doc/tutorial/

[13] GRIDCC D8.3 Project Final Report.

[14] Crossbow. Imote2 hardware reference manual. September, 2007.

[15] Open Geospatial Consortium (OGC) Website. http://www.opengeospatial.org/ogc

[16] Botts, M., Robin, A., Davidson, J., and Simonis, I. OpenGIS Sensor Web Enablement Architecture Document. OGC 06-021r1. 2006.

[17] Chu, Xingchen. Open Sensor Web Architecture**:** Core Services. Thesis of Master of Information Technology. Grid Computing and Distributed Systems Laboratory. Department of Computer Science and Software Engineering. University of Melbourne, Australia, 2005.

[18] 52North Website: http://52north.org

[19] Botts, Mike. OpenGIS Sensor Model Language (SensorML) Implementation Specification. OGC Draft 05-086. Open Geospatial Consortium. 2005.

[20] Cox, Simon (Ed). Observations and Measurements. OGC Draft 05-087r3. Open Geospatial Consortium. 24 February, 2006.

[21] Na, A., and Priest, M. Sensor Observation Service Implementation Specification. OGC 06-009r1. 2006.

[22] Simonis, I., OpenGIS Sensor Planning Service Implementation Specification. OGC 07-014r3. 2007.

Appendix I – Emulators/Simulators Comparison

Simulation Tool	TOSSIM	ns-2	GloMoSim (Qualnet)	OMNeT++ (Omnest)	OPNET
Simulator/Emulator	Simulator, Emulator	Simulator	Simulator	Simulator	Simulator
Transferability of Code	Yes	No	No	No	No
License (GPL, Commercial)	GPL	GPL	QualNet: Commercial	Omnest: Commercial	Commercial
Scalability	Yes - The same application code	No - n² relationship, every node is its own object	No	Yes	No
Architecture	Component-based	Object-oriented	Object-oriented	Component-based	Object-oriented
Platform	Windows, Linux	Windows, Linux, Sun, Mac	Windows, Linux, Sun, Mac	Windows, Linux, FreeBSD, Mac OS X	Linux. Windows
Popularity	Yes	Yes	Yes	No	No
WSN Platforms supported	MSP, AVR	No	No	No	No
Programming language of the tool / models	nesC / java(GUI)	C++ (Protocols), Otcl(Topology)	Parsec (C, C++)	C++, NED	C, C++
GUI, API etc.	TinyVIZ / SimDriver	GUI: nam	Yes	GUI: TkEnv	GUI
Parallel execution	Yes but not heterogeneous	Pdns	Parsec (C-based discrete event simulation language)	Yes: Parsim	Yes
Application Models (Hardware, Aplication and Network Protocol Level)	Discrete event queue, external programms	No		Modules to protocol layer, coordinator, protocol layer, Hardware Abstraction	Yes
Radio Propagation Model	Lossy Model, Empirical and Fixed Radius	Free space model, Two-ray ground model, Shadowing model and externals Realistic Channel and Ricean Propagation model			
Physical layer and antenna models	Empirical, Fixed Radius		SNR bounded, BER based with BPSK/QPSK modulation		Yes
Mobility models	No Mobility	Random Waypoint, Gauss-Markov, Manhattan Grid, Reference Point Group	RWP, Random Drunken, Trace based	No Mobility	Yes

Standards supported	802.15.4	AODV, OLSR, DYMO, 802.11, Bluetooth, Mobile IP	CSMA, IEEE 802.11, MACA, IP with AODV, Bellman-Ford, DSR,WRP, TCP, UDP, CBR, FTP, HTTP, Telnet and more	802.11	802.11, 802.16, MANET, MobileIP
Supports Energy Consumption Research	with extension PowerTOSSIM				
Statistical Support Tools	No	(ns2measure)	No	Akaroa RNG, seedtool	BSD RNG, Batch Means

Simulation Tool	**Avrora**	**J-SIM**	**SENSE**	**ATEMU**	**DISENS**
Simulator/Emulator	Simulator, Emulator	Simulator	Simulator	Simulator, Emulator	Simulator
Transferability of Code	Yes	No		Yes	
License (GPL, Commercial)	GPL	GPL	GPL	GPL	GPL
Scalability	No - n² reduced by syncronization	Yes	Yes (Reducing memory usage)	No - 120 Nodes - n²	Yes
Architecture	Emulate hw directly Object-oriented	Component-based	Component-based	Object-oriented (Overhead)	Discrete-Event
Platform	JVM	Linux, Windows	Linux, Windows (cgwin)	Linux	Linux
Popularity	No	No	No	No	No
WSN Platforms supported	AVR	No		AVR	
Programming language of the tool / models	Java	Java	C++ Templates	XML (Configuration, MICA2 Hardware)	
GUI, API etc.	No	No	No	GUI: XATDB (Debugger)	No
Parallel execution					Yes (Distributed)
Application Models (Hardware, Aplication and Network Protocol Level)			Power Model, Battery Model, Routing Protocols Models		Pluggable Models, Power Models, Radio Models...
Radio Propagation Model					
Physical layer and antenna models	Ideal	No			
Mobility models	No Mobility		No		
Standards supported	TinyOS MAC, any application	Only MAC in 802.11	802.11		
Supports Energy Consumption Research		Yes	No		
Statistical Support Tools	No				

Appendix II – An example of SOS commands response

SOS GetCapabilities response

```
<?xml version="1.0" encoding=" UTF-8"?>
<Capabilities xmlns="http://www.opengeospatial.net/sos" xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:swe="http://www.opengis.net/swe"
xmlns:om="http://www.opengis.net/om/1.0" xmlns:ows="http://www.opengeospatial.net/ows"
xmlns:ogc="http://www.opengis.net/ogc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengeospatial.net/sos:\ogc\svn\trunk\sos\0.0.31\sosGetCapabilities.xsd
http://www.opengis.net/om C:\ogc\svn\trunk\om\1.0.30\observation.xsd">
 <ows:ServiceIdentification>
  <ows:Title>SOS</ows:Title>
  <ows:Abstract>Example SOS Server</ows:Abstract>
  <ows:Keywords>
   <ows:Keyword>GERES Observation</ows:Keyword>
  </ows:Keywords>
  <ows:ServiceType codeSpace="http://opengeospatial.net">OGC:SOS</ows:ServiceType>
  <ows:ServiceTypeVersion>0.0.31</ows:ServiceTypeVersion>
 </ows:ServiceIdentification>
 <ows:ServiceProvider>
  <ows:ProviderName>Departamento de Informática - Universidade do Minho</ows:ProviderName>
  <ows:ProviderSite xlink:href="http://www.di.uminho.pt"/>
  <ows:ServiceContact>
   <ows:IndividualName>Luiz Fernando</ows:IndividualName>
   <ows:PositionName>Research Scientist</ows:PositionName>
   <ows:ContactInfo>
    <ows:Phone>
     <ows:Voice>967025746</ows:Voice>
     <ows:Facsimile>967025746</ows:Facsimile>
    </ows:Phone>
    <ows:Address>
     <ows:DeliveryPoint>Departamento de Informática - Universidade do Minho</ows:DeliveryPoint>
     <ows:City>BRAGA</ows:City>
     <ows:AdministrativeArea>Gualtar</ows:AdministrativeArea>
     <ows:PostalCode>4710</ows:PostalCode>
     <ows:Country>Portugal</ows:Country>
    </ows:Address>
   </ows:ContactInfo>
  </ows:ServiceContact>
 </ows:ServiceProvider>
 <ows:OperationsMetadata>
  <ows:Operation name="GetCapabilities">
 <ows:DCP>
  <ows:HTTP>
   <ows:Get xlink:href="http://www.dfrc.nasa.gov:8080/ogc2/GetCapabilities.ogc?"/>
   <ows:Post xlink:href="http://www.dfrc.nasa.gov:8080/ogc2/GetCapabilities.ogc"/>
  </ows:HTTP>
 </ows:DCP>
</ows:Operation>
<ows:Operation name="GetObservation">
 <ows:DCP>
  <ows:HTTP>
   <ows:Post xlink:href="http://www.dfrc.nasa.gov:8080/ogc2/GetObservation.ogc"/>
  </ows:HTTP>
 </ows:DCP>
 <ows:Parameter name="observedProperty" use="optional">
  <ows:Value>urn:x-ogc:def:phenomenon:OGC:AirTemperature</ows:Value>
  <ows:Value>urn:x-ogc:def:phenomenon:OGC:RelativeHumidity</ows:Value>
  <ows:Value>urn:x-ogc:def:phenomenon:OGC:Radiation</ows:Value>
  <ows:Value>urn:x-ogc:def:phenomenon:OGC:WindDirection</ows:Value>
  <ows:Value>urn:x-ogc:def:phenomenon:OGC:WindSpeed</ows:Value>
 </ows:Parameter>
</ows:Operation>
<ows:Operation name="DescribeSensor">
 <ows:DCP>
  <ows:HTTP>
   <ows:Post xlink:href="http://ren.3eti.net:8080/ogc2/DescribeSensor.ogc"/>
  </ows:HTTP>
 </ows:DCP>
 <ows:Parameter name="procedure" use="required">
  <ows:Value>urn:ogc:object:feature:Sensor:UM:um-sensor-north</ows:Value>
  <ows:Value>urn:ogc:object:feature:Sensor:UM:um-sensor-south</ows:Value>
 </ows:Parameter>
</ows:Operation>
</ows:OperationsMetadata>
<Contents>
<ObservationOfferingList>
 <ObservationOffering gml:id="GERES_NORTH_TEMPERATURE">
  <procedure xlink:href="urn:ogc:object:feature:Sensor:UM:um-sensor-north"/>
  <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:AirTemperature"/>
  <resultFormat>text/xml</resultFormat>
 </ObservationOffering>
 <ObservationOffering gml:id="GERES_NORTH_HUMIDITY">
  <procedure xlink:href="urn:ogc:object:feature:Sensor:UM:um-sensor-north"/>
  <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:RelativeHumidity"/>
 <resultFormat> text/xml</resultFormat>
</ObservationOffering>
<ObservationOffering gml:id="GERES_NORTH_RADIATION">
 <procedure xlink:href="urn:ogc:object:feature:Sensor:UM:um-sensor-north"/>
 <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:Radiation"/>
 <resultFormat> text/xml</resultFormat>
</ObservationOffering>
<ObservationOffering gml:id="GERES_NORTH_WIND_DIRECTION">
 <procedure xlink:href="urn:ogc:object:feature:Sensor:UM:um-sensor-south"/>
 <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:WindDirection"/>
 <resultFormat> text/xml</resultFormat>
</ObservationOffering>
<ObservationOffering gml:id="GERES_NORTH_WIND_SPEED">
 <procedure xlink:href="urn:ogc:object:feature:Sensor:UM:um-sensor-north"/>
 <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:WindSpeed"/>
 <resultFormat>text/xml</resultFormat>
</ObservationOffering>
<ObservationOffering gml:id="GERES_NORTH_WHEATER">
 <procedure xlink:href="urn:ogc:object:feature:Sensor:UM:um-sensor-north"/>
 <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:AirTemperature"/>
 <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:RelativeHumidity"/>
 <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:Radiation"/>
 <resultFormat>text/xml</resultFormat>
</ObservationOffering>
<ObservationOffering gml:id="GERES_SOUTH_TEMPERATURE">
 <procedure xlink:href="urn:ogc:object:feature:Sensor:UM:um-sensor-south"/>
 <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:AirTemperature"/>
 <resultFormat>text/xml</resultFormat>
</ObservationOffering>
<ObservationOffering gml:id="GERES_SOUTH_HUMIDITY">
 <procedure xlink:href="urn:ogc:object:feature:Sensor:UM:um-sensor-south"/>
 <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:RelativeHumidity"/>
 <resultFormat> text/xml</resultFormat>
</ObservationOffering>
<ObservationOffering gml:id="GERES_SOUTH_RADIATION">
 <procedure xlink:href="urn:ogc:object:feature:Sensor:UM:um-sensor-south"/>
 <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:Radiation"/>
 <resultFormat> text/xml</resultFormat>
</ObservationOffering>
<ObservationOffering gml:id="GERES_SOUTH_WIND_DIRECTION">
 <procedure xlink:href="urn:ogc:object:feature:Sensor:UM:um-sensor-south"/>
 <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:WindDirection"/>
 <resultFormat> text/xml</resultFormat>
```

```
   </ObservationOffering>
   <ObservationOffering gml:id="GERES_SOUTH_WIND_SPEED">
    <procedure xlink:href="urn:ogc:object:feature:Sensor:UM:um-sensor-south"/>
    <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:WindSpeed"/>
    <resultFormat>text/xml</resultFormat>
   </ObservationOffering>
   <ObservationOffering gml:id="GERES_SOUTH_WHEATER">
    <procedure xlink:href="urn:ogc:object:feature:Sensor:UM:um-sensor-south"/>
    <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:AirTemperature"/>
    <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:RelativeHumidity"/>
    <observedProperty xlink:href="urn:x-ogc:def:phenomenon:OGC:Radiation"/>
    <resultFormat>text/xml</resultFormat>
   </ObservationOffering>
  </ObservationOfferingList>
 </Contents>
</Capabilities>
```

SOS DescribeSensor response

```
<?xml version="1.0" encoding=" UTF-8"?>
<sml:SensorML xsi:schemaLocation="http://www.opengis.net/sensorML/1.0.1
http://schemas.opengis.net/sensorML/1.0.1/sensorML.xsd" version="1.0.1" xmlns="http://www.opengis.net/sensorML/1.0.1"
xmlns:swe="http://www.opengis.net/swe/1.0.1" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
xmlns:gml="http://www.opengis.net/gml">
 <sml:identification>
  <sml:IdentifierList>
   <sml:identifier name="URN">
    <sml:Term definition="urn:ogc:def:identifierType:OGC:uniqueID">
     <sml:value>urn:ogc:object:feature:Sensor:UM:um-sensor-north</sml:value>
    </sml:Term>
   </sml:identifier>
   <sml:identifier name="longName">
    <sml:Term>
     <sml:value>UM Sensor North</sml:value>
    </sml:Term>
   </sml:identifier>
   <sml:identifier name="shortName">
    <sml:Term>
     <sml:value>sensor north</sml:value>
    </sml:Term>
   </sml:identifier>
   <sml:identifier name="modelNumber">
    <sml:Term>
     <sml:value>imote2</sml:value>
    </sml:Term>
   </sml:identifier>
   <sml:identifier name="manufacturer">
    <sml:Term>
     <sml:value>crossbow</sml:value>
    </sml:Term>
   </sml:identifier>
  </sml:IdentifierList>
 </sml:identification>
 <sml:classification>
  <sml:ClassifierList>
   <sml:classifier name="intendedApplication">
    <sml:Term>
     <sml:value>AirTemperature</sml:value>
    </sml:Term>
   </sml:classifier>
   <sml:classifier name="intendedApplication">
    <sml:Term>
     <sml:value>RelativeHumidity</sml:value>
    </sml:Term>
   </sml:classifier>
   <sml:classifier name="intendedApplication">
    <sml:Term>
     <sml:value>Radiation</sml:value>
    </sml:Term>
   </sml:classifier>
   <sml:classifier name="intendedApplication">
    <sml:Term>
     <sml:value>WindDirection</sml:value>
    </sml:Term>
   </sml:classifier>
   <sml:classifier name="intendedApplication">
    <sml:Term>
     <sml:value>WindSpeed</sml:value>
    </sml:Term>
   </sml:classifier>
   <sml:classifier name="sensorType">
    <sml:Term>
     <sml:value>AirTemperature</sml:value>
    </sml:Term>
```

```
</sml:classifier>
<sml:classifier name="sensorType">
 <sml:Term>
  <sml:value>RelativeHumidity</sml:value>
 </sml:Term>
</sml:classifier>
<sml:classifier name="sensorType">
 <sml:Term>
  <sml:value>Radiation</sml:value>
 </sml:Term>
</sml:classifier>
<sml:classifier name="sensorType">
 <sml:Term>
  <sml:value>WindDirection</sml:value>
 </sml:Term>
</sml:classifier>
<sml:classifier name="sensorType">
 <sml:Term>
  <sml:value>WindSpeed</sml:value>
 </sml:Term>
</sml:classifier>
<sml:classifier name="phenomenon">
 <sml:Term>
  <sml:value>AirTemperature</sml:value>
 </sml:Term>
</sml:classifier>
<sml:classifier name="phenomenon">
 <sml:Term>
  <sml:value>RelativeHumidity</sml:value>
 </sml:Term>
</sml:classifier>
<sml:classifier name="phenomenon">
 <sml:Term>
  <sml:value>Radiation</sml:value>
 </sml:Term>
</sml:classifier>
<sml:classifier name="phenomenon">
 <sml:Term>
  <sml:value>WindDirection</sml:value>
 </sml:Term>
</sml:classifier>
<sml:classifier name="phenomenon">
 <sml:Term>
```

```
    <sml:value>WindSpeed</sml:value>
   </sml:Term>
  </sml:classifier>
 </sml:ClassifierList>
</sml:classification>
<sml:member>
 <sml:System gml:id="um-sensor-north">
  <sml:position name="actualPosition"> <!-- GeometryFromText('POINT(7.727958 71.883906)', 4326) -->
   <swe:Position fixed="false" referenceFrame="urn:ogc:crs:epsg:4326">
    <swe:location>
     <swe:Vector>
      <swe:coordinate name="x">
       <swe:Quantity>
        <swe:value>7.727958</swe:value>
       </swe:Quantity>
      </swe:coordinate>
      <swe:coordinate name="y">
       <swe:Quantity>
        <swe:value>71.883906</swe:value>
       </swe:Quantity>
      </swe:coordinate>
     </swe:Vector>
    </swe:location>
   </swe:Position>
  </sml:position>
  <sml:inputs>
   <sml:InputList>
    <sml:input name="temperature">
     <swe:ObservableProperty definition="urn:x-ogc:def:phenomenon:OGC:AirTemperature"/>
    </sml:input>
    <sml:input name="humidity">
     <swe:ObservableProperty definition="urn:x-ogc:def:phenomenon:OGC:RelativeHumidity"/>
    </sml:input>
    <sml:input name="radiation">
     <swe:ObservableProperty definition="urn:x-ogc:def:phenomenon:OGC:Radiation"/>
    </sml:input>
    <sml:input name="windDirection">
     <swe:ObservableProperty definition="urn:x-ogc:def:phenomenon:OGC:WindDirection"/>
    </sml:input>
    <sml:input name="windSpeed">
     <swe:ObservableProperty definition="urn:x-ogc:def:phenomenon:OGC:WindSpeed"/>
    </sml:input>
   </sml:InputList>
```

```
    </sml:inputs>
    <sml:outputs>
     <sml:OutputList>
      <sml:output name="temperatureMeasurements">
       <swe:Quantity definition="urn:x-ogc:def:phenomenon:OGC:AirTemperature">
        <swe:uom code="urn:x-ogc:def:unit:degC"/>
       </swe:Quantity>
      </sml:output>
      <sml:output name="humidityMeasurements">
       <swe:Quantity definition="urn:x-ogc:def:phenomenon:OGC:RelativeHumidity">
        <swe:uom code="urn:x-ogc:def:unit:percent"/>
       </swe:Quantity>
      </sml:output>
      <sml:output name="RadiationMeasurements">
       <swe:Quantity definition="urn:x-ogc:def:phenomenon:OGC:Radiation">
        <swe:uom code="urn:x-ogc:def:unit:cd"/>
       </swe:Quantity>
      </sml:output>
      <sml:output name="WindDirectionMeasurements">
       <swe:Quantity definition="urn:x-ogc:def:phenomenon:OGC:WindDirection">
        <swe:uom code="urn:x-ogc:def:unit:degree"/>
       </swe:Quantity>
      </sml:output>
      <sml:output name="WindSpeedMeasurements">
       <swe:Quantity definition="urn:x-ogc:def:phenomenon:OGC:WindSpeed">
        <swe:uom code="urn:x-ogc:def:unit:meterPerSecond"/>
       </swe:Quantity>
      </sml:output>
     </sml:OutputList>
    </sml:outputs>
   </sml:System>
  </sml:member>
</sml:SensorML>
```

SOS GetObservation response

```
<?xml version="1.0" encoding=" UTF-8"?>
<om:Observation xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:gml="http://www.opengis.net/gml"
xmlns:om="http://www.opengis.net/om/1.0"  xmlns:swe="http://www.opengis.net/swe"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/om/1.0
http://schemas.opengis.net/om/1.0.0/om.xsd">
  <gml:name>urn:UM:observation:AirTemperature</gml:name>
```

```
<om:samplingTime>
 <gml:TimePeriod>
  <gml:beginPosition> 2008-10-06T10:15:00Z </gml:beginPosition>
  <gml:endPosition> 2008-10-06T10:20:00Z </gml:endPosition>
 </gml:TimePeriod>
</om:samplingTime>
<om:procedure xlink:href="urn:ogc:object:feature:Sensor:UM:um-sensor-north"/>
<om:observedProperty xlink:href="urn:ogc:def:property:OGC:AirTemperature"/>
<om:featureOfInterest xlink:href="id_N"/>
<om:result>
 <swe:DataArray gml:id="AirTemperature">
  <swe:elementCount>
   <swe:Count>
    <swe:value>6</swe:value>
   </swe:Count>
  </swe:elementCount>
  <swe:elementType name="Components">
   <swe:SimpleDataRecord gml:id="DataDefinition">
    <swe:field name="time">
     <swe:Time definition="urn:ogc:property:time:iso8601"/>
    </swe:field>
    <swe:field name="AirTemperature">
     <swe:Quantity definition="urn:ogc:def:property:OGC:AirTemperature">
      <swe:uom code="degC"/>
     </swe:Quantity>
    </swe:field>
   </swe:SimpleDataRecord>
  </swe:elementType>
  <swe:encoding>
   <swe:TextBlock tokenSeparator="," decimalSeparator="." blockSeparator="@@"/>
  </swe:encoding>
  <swe:values> 2008-10-06T10:15:00Z,26.0@@2008-10-06T10:16:00Z,26.3@@
  2008-10-06T10:17:00Z,26.2@@2008-10-06T10:18:00Z,26.0@@
  2008-10-06T10:19:00Z,26.1@@2008-10-06T10:20:00Z,26.0@@
  </swe:values>
 </swe:DataArray>
</om:result>
</om:Observation>
```