An OGC/SOS Conformant Client to Manage Geospatial Data on the GRID

António Esteves[1], Marco Caldas[1], António Pina[1], Alberto Proença[1]

[1] Informatics Department, University of Minho, Braga, Portugal
{esteves,marcocaldas,pina,aproenca}@di.uminho.pt

Abstract. This paper describes a Sensor Observation Service (SOS) client developed to integrate dynamic geospatial data from meteorological sensors, on a grid-based risk management decision support system. The present work is part of the CROSS-Fire project, which uses forest fires as the main case study and the FireStation application to simulate fire spread. The meteorological data is accessed through the SOS standard from Open Geospatial Consortium (OGC), using the Observations and Measurements (O&M) standard encoding format. Since the SOS standard was not designed to directly access sensors, we developed an interface application to load the SOS database with observations from a Vantis Weather Station (WS). To integrate the SOS meteorological data into the FireStation, the developed SOS client was embedded on a Web Processing Service (WPS) algorithm. This algorithm was designed to be functional and fully compliant with SOS, SensorML, and O&M standards from OGC. With minor modifications to the developed SOS database interface, the SOS client works with any WS. This client supports spatial and temporal filters, including the integration of dynamic data from satellites into FireStation, as described.

Keywords: OGC stantards, SOS, WPS, geospatial data, grid civil protection applications.

1 Introduction

The CROSS-Fire project aims to develop a grid-based risk management decision support system, using the Enabling Grids for E-sciencE (EGEE) infrastructure, for the Civil Protection (CP) authorities, using forest fires as the main case study and FireStation (FS) as a standalone CAD application to simulate the fire spread over complex topography [12]. The CROSS-Fire architecture includes information models, encodings, and metadata that represent the scientific knowledge associated to FireStation execution models and standards to enable the discovery and access of Web services, data repository, sensor networks and data processing facilities [16]. To achieve the desired integration of information and services we use: (i) EGEE to provide raw technological capability provision, including data management and storage, access to meta-data data bases and high-performance computing and (ii) a Geospatial Information Infrastructure based on OGC-WS and Sensor Web Enablement (SWE) services to provide the access and management of remote geospatial data from remote or in-situ sensors.

1.1 FireStation and G-FireStation

FireStation integrates a module for wind field generation, as well as a module for the computation of the Fire Weather Index (FWI). FS needs three different kinds of input data to simulate fire propagation: (i) the terrain divided into cells, each one characterized by its altitude and fuel type [13], (ii) the wind conditions, affecting that terrain and (iii) some control parameters, such as the ignition points and the stopping simulation criteria (see figure 1). The information about the wind conditions affecting the terrain is previously generated by a Wind Field Module.

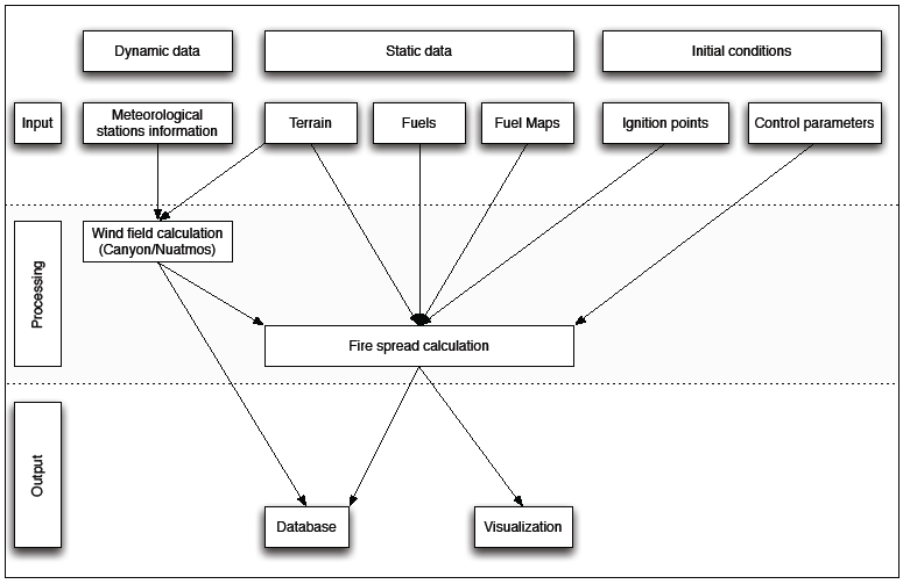

Fig. 1: Components of FS Simulation model.

The FireStation is being ported to work on grid, resulting in the G-FireStation (gFS) application. G-FireStation includes: (i) a standard-based Spatial Data Infrastructure (SDI) layer, based on Geoserver, to exploit/enable geospatial services for data access/processing, (ii) a 52North's implementation of a OGC/SWE conformant layer, to address sensors CP data sources, such as meteorological stations data and satellite images and (iii) a graphical user interface to access the platform facilities.

To provide FireStation with dynamic geospatial and meteorological data needed for computing the wind field (wind direction and speed) and FWI (precipitation, temperature, and humidity) we are developing a Web-interface for Remote Weather Stations (RWS). The implementation is based on a Sensor Observation Service (SOS) server from 52North that manages the access to a database that conforms to SOS from OGC/SWE initiative. The Web-interface is being tested with a Davis VantagePro2 Weather Station (WS). A simple SOS client retrieves

the data from a RWS that is connected to a FTP server, performs the necessary processing and uses an *insertObservation* client operation to insert data in the SOS database. The functionality of the SOS client is used to implement a Web Processing Service (WPS) algorithm that runs as a core service of the CROSS-Fire. Concretely, the algorithm is be based on the developed *GetObservation* class that allows WPS (i) to request, from the SOS, the list of WSs available on a certain spatial window, and (ii) to retrieve observations from sensors belonging to a specific station included on the previous list.

1.2 Cross-Fire Platform

The CROSS-Fire platform is composed of a central core, a WPS layer, and two external infrastructures: a SDI platform and the GRID. The core WPS, a 52North implementation of this OGC standard layer, is divided in three parts: Business Logic, Grid Services and Geospatial services. The Business Logic is an abstract layer configured to handle the specific algorithms that provide all the functionality of FireStation, namely forest fire propagation, wind field and FWI calculation. The Grid Services is the component that interfaces with the GRID infra-structure. Amongst its responsibilities one can find proxy delegation, job creation and management, and data movement to and from the GRID. The Geospatial Services act as an interface between the clients who request geo-referenced data and the available collection of SDIs. Web services are one of the fundamental layers of the CROSS-Fire platform, implemented as a set of WPS algorithms that deal with most of the functionalities of the three components of the platform.

The paper is organized as it follows. In section 2 we summarize some available clients for the 52North implementation of the SOS. Section 3 describes the integration of meteorological dynamic data on Firestation. It is presented an overview of OGC/SWE standards, especially the Sensor Observations Service and the Observations&Measurements. In section 4 it is given a brief description of the weather station used to test the SOS. Section 5 details the application developed to populate the SOS database with observations from this weather station. Section 6 describes an SOS client implementation. This client allows WPS to retrieve geo-referenced spatial data from SOS. Finally, section 7 points out some conclusions and ideas for future work.

2 Related Work

There are available several client applications for the 52North's implementation of the SOS service [1]. Most of them are implemented on top of the OX-Framework that facilitates the access to OGC web services.

The `Rich OX client` is a Java Swing frontend of the OX-Framework. Its focus is on the visualization of O&M-encoded sensor data requested from an SOS. This client also interfaces with WMS, WCS and SAS services, offering basic GIS functionalities such as: display information on a map, as a diagram, or as an animation, manage map layers, zooming and panning of a spatial area, perform interpolation over the available data and display the result on a map.

The `Thin SWE client` is a web-based client with a user friendly interface. This client visualizes sensor data on maps, diagrams and tables. In order to be a platform independent application, it was developed as a web-based thin client, runs on a browser and uses web technologies like AJAX. The user can access data from different SOS instances. Besides giving access to sensor observations, this application also presents metadata about the sensor.

The `ArcGIS SOS extension` couples SOS instances to ArcGIS. With this extension, the users can apply the full range of ArcGIS processing and visualization capabilities to real-time and stored sensor data. The `ArcGIS extension` requires the ArcGIS and Arc Hydro tools.

The `uDig SOS client` is a GIS application written in Java. `uDig` is a free and open source project. This client allows the `uDig` system to access SOS data. The information retrieved from the SOS can be displayed on a map, using a `uDig` layer, or as a table. It is also possible to perform data rendering. In the present stage of development, the client does not implement temporal filtering on data.

The Commonwealth Scientific and Industrial Research Organization (CSIRO), an Australia's national science agency, is investigating how OGC/SWE standards can be applied to the hydrological domain, especially to the South Esk river. They also developed a SOS client to access the sensor's observations and a Google Maps GUI to display the localization of the available sensors.

3 Integrating Meteorological Dynamic Data on Firestation

3.1 OGC/SWE Standards

To access the weather information it was decided to adopt the standards developed by the *Sensor Web Enablement* (SWE) initiative of the OGC. Concretely, we based our solution on the most commonly used implementation on the scientific community, and which was developed by the 52North company [19]. The SWE initiative provides models, services and encoding schema that allow an access to web sensors through a common interface and encoding. The objective of this initiative is the standardization of all the process of accessing web sensors, which includes: the description of sensors, the register of sensors and their measurements, the discovery of sensors and how to interact with them, and the access to the observations provided by sensors. To achieve this objective, the SWE initiative defined 3 standards for data encoding and 4 standards for interaction with web services: SensorML [4], *Observations&Measurements* (O&M) [6, 7], *Transducer Markup Language* (TML), and *Sensor Observations Service* (SOS) [14], *Sensor Planning Service* (SPS) [18], *Sensor Alert Service* (SAS), *Web Notification Service* (WNS).

We can setup various scenarios for interaction between SWE services and encodings, some simple and others quite complex. In the scenario we want to explore in CROSS-Fire, the user (a Civil Protection application, i.e. the WPS) is not usually interested in all observations generated by the sensors, only the observations associated with specific situations. For example, we just want to know when the wind direction, or speed, had a significant change. In this case, the user wants to be notified by an SAS when this change occurs (figure 2).

To implement a interaction scenario of this kind, the user searches, on a catalogue service, for SAS services that meet the requirements of scenario in question. Later, it can connect to one of the SAS found, in order to be notified when the mentioned change occurs. SAS replaces the SOS in the task of monitoring continuously the sensor(s). This means that the sensor registers and publishes the observations into the SAS. The best way to implement this kind of asynchronous interaction, is using an SPS. Then, the user must also connect to an SPS and schedule on it the task appropriate to the scenario in question. For example, the task can specify that when occurs a significant change in wind, the WNS should be notified. After this occurrence, the WNS is responsible for forwarding the notification to the user. More or less at the same time, the user also receives a message notification from SAS. If the user wants to know the value of the observed property (wind direction or wind speed), at the time of the notification, he has to connect first to an SOS and send him a *getObservation* request after receiving the notification. The answer will be an O&M document containing the the wind direction, or wind speed, observation.

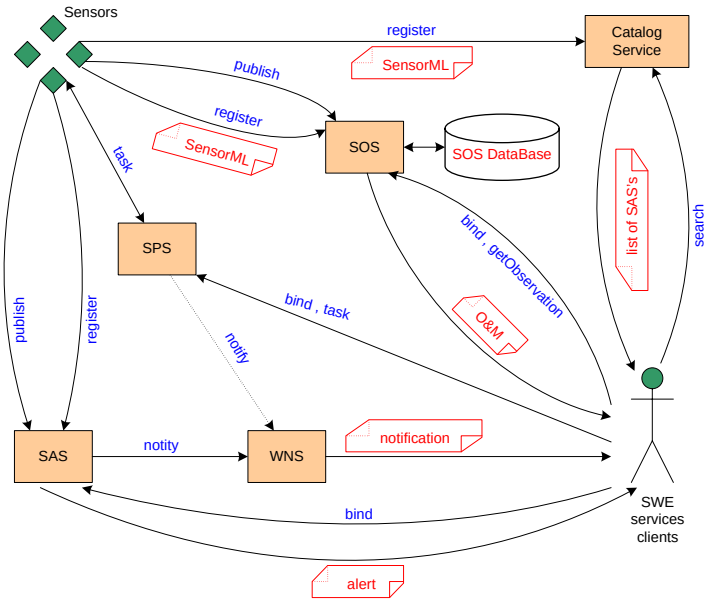

Fig. 2: Scenario of SWE services interaction.

3.2 Sensor Observations Service

Since the *Sensor Observations Service* (SOS) [14] is the nuclear service to access sensors, is the one that will be treated in more detail in this document. SOS defines a standard web interface to request, filter, and get observations and metadata from sensors.

The *getCapabilities* is a mandatory operation used for receiving metadata about the potentialities of the service. A *getCapabilities* request should include the type of service that apply ("SOS" in this case), and which sections of the *capabilities document* we are interested on. The answer to a *getCapabilities* request is an XML document containing metadata about the potentialities of the service, namely which operations are implemented and the location of the SensorML description of each sensor.

The mandatory *describeSensor* operation allows us to access information of each (local or remote) sensor, returning a SensorML or TML document. A *describeSensor* request must specify the format used in output data, the sensor identification, the service identification and version, and the procedure needed to obtain observations from that sensor, which is nothing more than the sensor Uniform Resource Name (URN). The answer to *describeSensor* request is a SensorML or TML document describing the sensor. A summary of what one can get in a SensorML document is: (i) the location of the sensor and (ii) the phenomena that sensor monitors.

The *getObservation* is a mandatory operation used to request observations, encoded in a O&M document. A *getObservation* request, containing all the fields specified in the response to a *getCapabilities*, includes: the service and its version, the *offering* we are interested in, the desired sampling time, the procedure, the observed property, the *feature of interest*, the observation result, the model used on the result, and the type of answer. The response to a *GetObservation* request is usually an O&M document. The simplified structure of an observation encoded in O&M [6], presented in figure 3, relates the observation with the feature of interest, the applied procedures, the observed property, the result and the quality of the result value, the time at which the sample was collected and the time the result was generated (which may be equal to the previous time), some additional metadata and parameters describing the event that resulted in the observation but are neither strongly connected to the process nor to the *feature of interest*.

The *registerSensor* is an optional operation, which allows the SWE client to register a sensor on SOS. The client can only insert observations belonging to a sensor already registered with the SOS. A *registerSensor* request registers the sensor in the SOS and returns the ID assigned to it. This ID should be used to identify unequivocally the sensor when we interact with the SOS, for example when making a *describeSensor* request.

The *insertObservation* operation allows an SOS client to insert observations in the system associated with a sensor. The client needs to indicate the ID of the sensor that produced the observation and the observation has to be specified through an O&M document. The answer to a *insertObservation* request is the ID of the observation that has just been inserted, which can be used later to request the observation in question, sending a *getObservationById* request to the SOS.

4 The Weather Station Used to Test SOS

The weather station used to test the SOS was a Davis Vantage Pro2 [9]. This weather station consists of a base station and a sensor suite. Among the available

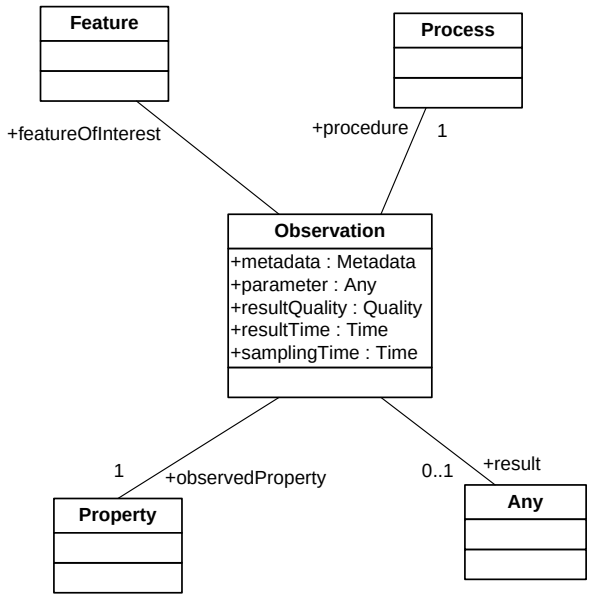

Fig. 3: Simplified structure of an O&M observation.

sensors, we should mention a rain collector, a temperature sensor, a humidity sensor, an anemometer, a barometer, wind speed and wind direction sensors. These are the relevant sensors to Firestation's fire propagation model. The accuracy of the sensors included on the weather station is as follows: +/- 5% for the rain collector, +/- 0.5^oC for the temperature sensor, +/- 3% for the humidity sensor, +/- 5% for the anemometer and +/- 4^o for the wind direction sensor. The measured values are sent through a wireless connection to the base station, yet distance can not exceed 150 meters. Because of the base station can not be linked directly to a PC, it is necessary an additional *logger* with a capacity for 2560 records. The *logger* can be connected to a PC via USB. The communication protocol between *logger* and PC is available freely.

5 The Application that Populates the SOS Database

To access the remote Vantis weather station, which is connected to an FTP server, it was developed an application that connects to that server in order to obtain meteorological data. This data is later processed and inserted in the SOS database. It was used a PostgreSQL database. The application was implemented in JAVA language, since it is multi-platform. The station in question is the Davis Vantage Pro2 described in section 4, which can be configured through a template that defines the meteorological fields that must be saved, the FTP site to where the data will be periodically sent, as well as the format of the generated data file (TXT, HTML, XML, etc). In order to facilitate the processing of the data file it

was selected the XML format. To insert the information in the SOS database, it was necessary to implement a client version of the SOS *insertObservation* operation. This operation will receive the meteorological data, which have to be embedded in an request that adheres to the format and structure defined by the standard. In particular, the *insertObservation* request must be a valid XML document and include fields that identify: (i) the sensors involved in this observation, (ii) the date or period in which the observation was collected, (iii) the address of the applied procedure, (iv) the observed phenomena, (v) the coordinates of the location where observation was collected, (vi) the set of values of the observed phenomena, properly identified and structured.

In order to insert information in the SOS database, for example information about wind, it is necessary to have registered previously the wind sensor on SOS. In this case, the wind sensor provides two metrics, wind direction and wind speed. To register sensors it is used the SOS *registerSensor* operation. A *registerSensor* request must be a valid XML document and contain fields that specify: (i) the sensor identifier, which must be unique for each SOS, (ii) the sensor state, which can be active/inactive and be fixed/mobile, (iii) the sensor position, i.e., the coordinates of its location, and (iv) the phenomena, with the respective units, that sensor monitors.

Once a sensor has been registered in the SOS, it is available to insert data relative to the phenomena it monitors. In order to execute this task automatic and transparently, it was developed the aforementioned Java application. This application connects to the FTP server, gets the last observation placed there by the weather station and saves locally a copy of this observation, properly identified with date and time. To be self-configurable, the application reads a configuration file at startup, defining values for the following parameters:

- The identification of the FTP server (IP, username and password);
- The name of the files that we want to download, and if they are to be processed or not;
- The way the application will work, including the frequency of the access to the FTP server, if it is required to save locally a copy of files downloaded from the FTP server and where they will be saved;
- Then it follows a section for the list of tags, providing from the file generated by the weather station, that we intend to process and insert into the SOS database. The section associated with each tag was includes:(i) the sensor identification, (ii) the tag name in the file generated by the meteorological station, (iii) when necessary, the factor to be applied on unit conversion, (iv) the unit, (v) the phenomenon under study, which must be listed in the SOS, (vi) the URN of the SOS phenomenon, (vii) the position where the sample was collected, i.e., the coordinates of its location.

The application consists of three parsers: (i) one for configuration file, (ii) another for the file generated by the weather station, which is conditioned by the first file, because the fields are described in the configuration file, and finally, (iii) a parser for the SOS answer. The application still includes a log file where it saves all the errors it generates. Figure 4 presents an illustration of the architecture

where the developed application fits. The data flow within the application includes: (i) to connect to the FTP server containing meteorological observations, (ii) to make a copy of the last observation, which will be subsequently used to look for the fields defined in the configuration file, (iii) to generate an XML document with the *insertObservation* request, containing the data from the observation in question, (iv) to send the request to the SOS and (v) to wait for the response to this request. If the request succeeds, the observation is inserted into the SOS database, otherwise an error is registered in the log file. It was also implemented a notification mechanism, via e-mail, to inform when the application crashes or is unable to obtain data from FTP. The notification system is quite simple. When 5 internal similar errors are reported consecutively, signaling that 5 consecutive samples were not inserted into the SOS, it is then issued an e-mail informing about the incident.

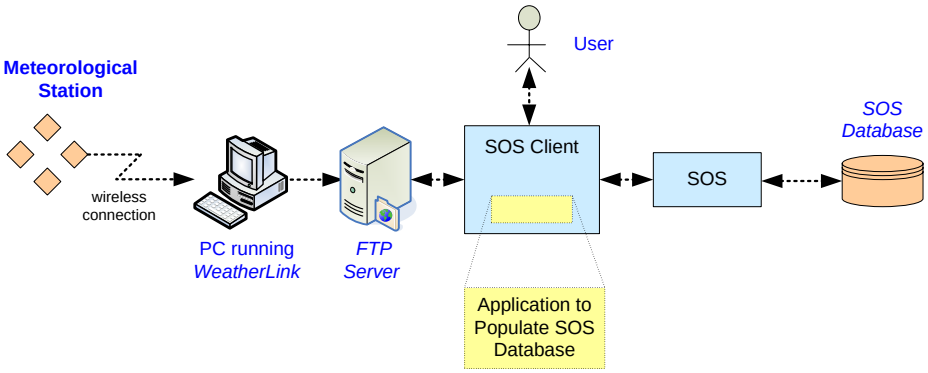

Fig. 4: Configuration of the meteorological station access through SOS.

6 Case Study: a WPS Algorithm that Accesses Geo-referenced Spatial Data

This section describes the facilities required by a WPS algorithm that deals with geo-referenced spatial data stored on a SOS database. In other words, it will be described an SOS client implementation to mediate the interaction between WPS [17] and SOS [14]. The SOS client utilization will be embedded on a WPS algorithm.

WPS is a OGC standard used to make calculations in a standard way through the Internet. In practice it is a web service. The mandatory operations provided by WPS are the following.

getCapabilites - Returns an XML document containing the name of the available algorithms, as well as other information about the WPS potentialities.

DescribeProccess - Returns details about a certain algorithm, including a brief description of what it does and the number and type of inputs and outputs.

Execute - Executes a specified algorithm and returns the results. In CROSSFire, where we use the WPS implementation from 52North, an execution request

is sent through an XML document that contains the name of the algorithm to be executed, its inputs and outputs.

At this point, the facilities necessary for the mentioned WPS algorithm are implemented as a *GetObservation* class, which provides two methods: (i) *getData* that allows WPS to consult the SOS database and (ii) *getStations* that lets WPS to know which weather stations are located in a given region. To implement this class it was necessary to develop the classes *CreateDoc*, *GetHTTP* and *ObservationOffering*, which will be described next.

CreateDoc - This internal class is responsible for creating the XML document to be sent as a request to the SOS. When the class is instantiated it creates a *StringBuffer* where it is inserted the header of the XML document to be sent to the SOS, then providing methods to add the several fields:

(i) *addOffering*, to add a weather station;
(ii) *setEventTime*, which receives the time operator (which is fixed and with value *TM_During*) and the initial and final dates;
(iii) *addProcedure*, to add the URN of the sensor to the document;
(iv) *addObservedProperty*, to add the feature of interest;
(v) *addBottom*, to add the footer part of the XML document;
(vi) *getDoc*, to return the XML document created.

GetHTTP - This class allows us to send requests to the SOS and receive the answer. It has a single method, *send*, which sends the text received as parameter to a certain URL. The implementation of the method is based on the external library "Jakarta HttpClient", which deals with all the tasks involved in the HTTP request.

ObservationOffering - This is a support class to store the data that will be returned by the *getStations* method from *GetObservation* class, described next. The class only contains methods for reading and writing the several fields, and that are: (i) *name*, the station name, (ii) *procedure*, the sensor URN and (iii) *observedProperty*, the list of phenomena observed by the meteorological station.

The two methods of the *GetObservation* class are now described.

getData - This method receives as parameters: (i) a list of offerings, which are the weather stations available in SOS, (ii) the initial and final dates, defining a timing window that delimitates the search, (iii) an identifier of the sensor in question, and finally (iv) the list of features of interest we are looking for, such as temperature, humidity or wind. The values needed to fill these parameters provide from the *getStations* method that, beyond informing which stations are located in a given area, returns additional information about each of them.

Firstly, the method creates an instance of the *CreateDoc* class that is responsible for creating the XML document to be sent as a request to the SOS. After the instance is created, it goes through the various attributes received in the method, and calls the methods of the *CreateDoc* class to insert them in the document. After the document is created and filled, it is sent to the SOS, using the *GetHTTP* class. The SOS answer, will be returned by a method of the *GetHTTP* class, which in turn returns it to the *getData* method that initiated the operation. When the data is in this *getData* method, it will be processed by a parser and placed in a *HashMap*

to be returned to the WPS algorithm that deals with geo-referenced spatial data, and called the *getData* method.

getStations - This method receives two coordinates and the identifier of the Spatial Referencing System (SRS) that will specify the Coordinate Reference System (CRS) to use, based on the the EPSG standard [15]. Initially, it will sent a *GetCapabilities* request to the SOS, to tell us which stations it offers, including its coordinates, which phenomena are observed by these stations and their URN. This information, embedded in the *Capabilities* document, will be processed by a parser to strip each of the fields mentioned above. In addition, it is applied a spatial filter in order to consider only stations that are within the specified region. This data will be stored in the *ObservationOffering* class that will be returned by the *getStations* method to the caller WPS algorithm.

Just for validation purposes, we decided to implement a graphical user interface (GUI) to the WPS algorithm (see figure 5). An example of operation with this GUI is now presented. Firstly, it is necessary to select a CRS, then we must enter two location points that define the area of interest. When the `GetStation` is pressed, the SOS database will be queried and the list of weather stations (WS), located in the selected area, will be displayed in the `Station` window. When we select a certain WS, the sensors that are available on this WS will be displayed in the `Observed property` window. After that, we must choose one or more sensors, introduce a time interval, and press `Get Observation` to display the observations stored in the SOS database (for that WS, sensor(s), and time interval).

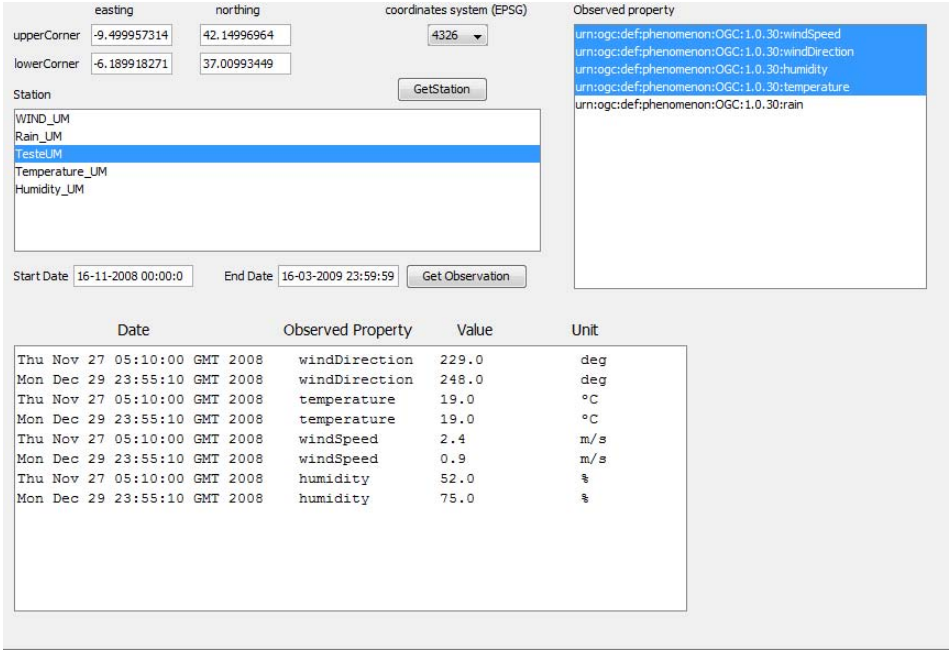

Fig. 5: An implementation of a GUI to the WPS algorithm.

7 Conclusions and Future Work

The main reason to implement a new SOS client, instead of using an available one, was the necessity of interacting with WPS, a facility not present on the reviewed clients. Those clients also offer GIS facilities that are not relevant in our case. The presented work includes the development of an SOS client, an application that populates the SOS database with data from a weather station (WS), and a WPS algorithm to access SOS and request from it observations from wind, temperature, humidity, and precipitation sensors. The developed SOS client is functional and fully compliant with SOS, SensorML, and O&M standards from OGC. With a slight modifying of the application that populates the SOS database, to take into account the specificities of the way observations are provided by the concrete WS, the implemented client works with any WS, without making any changes. The client supports spatial and temporal filters. In relation to the integration of meteorological data on Firestation it is necessary to finish the SOS client, concretely implement the parser for the *describeSensor* and *registerSensor* operations. It is our objective to integrate the SOS client on Google Maps to allow an easy localization of the weather stations accessible through the SOS.

We are also working on the integration of other types of spatial data, such as satellite images, on the CROSS-Fire project. To fulfill this task, projects like SAFORAH [5], Sentinel Asia [10], GEO Grid [22], RGI [11], MRR and FIRMS [8] were analyzed. These projects work with data from Terra, Aqua, EO1, and ALOS satellites, and sensed by MODIS [20], ASTER [2] and PALSAR instruments. Given the potential and the large community that uses data from Terra and Aqua, it is our intention to use the information from these satellites too. Specifically, we consider the utilization of the MODIS instrument an adequate alternative. The data we are interested in is mainly land coverage (vegetation) and burned areas to validate the maps of vegetation used by Firestation and the results of fire spread simulations. The information will be provided as coverages by a WCPS service [3], an extension of the WCS [21].

Acknowledgements

This research was funded by the Portuguese organization *Fundação para a Ciência e a Tecnologia* (FCT) through the CROSS-Fire project, a project with reference GRID/GRI/81795/2006. This research also includes results from the *EC FP7 EELA-2, E-science grid facility for Europe and Latin America: Deployment of e-Infrastructures for scientific communities* project.

References

1. 52North. "SWE Clients", `http://52north.org/maven/project-sites/swe/clients/`. 2010.
2. M. Abrams, S. Hook, and B. Ramachandran, "ASTER User Handbook, Version 2", *Jet Propulsion Laboratory, California Institute of Technology*, `http://asterweb.jpl.nasa.gov/documents.asp`, august (2002).

3. Peter Baumann. "The OGC Web Coverage Processing Service (WCPS) Standard", *Geoinformatica*, Springer, july (2009).
4. Mike Botts. "OpenGIS Sensor Model Language (SensorML) Implementation Specification", *Open Geospatial Consortium Document 07-000*, july (2007).
5. H. Chen, D, Goodenough, L. Di, A. Guan, A. Dyk and G. Hobart. "Grid-enabled OGC Environment for EO Data and Services in Support of Canada's Forest Applications", september (2007).
6. Simon Cox. "Observations and Measurements - part 1 - Observation Schema", *Open Geospatial Consortium Document 07-022r1*, december (2007).
7. Simon Cox, "Observations and Measurements - part 2 - Sampling Features", *Open Geospatial Consortium Document 07-002r3*, december, (2007).
8. D. Davies, S. Ilavajhala, M. Wong and C. Justice. "Fire Information for Resource Management System: Archiving and Distributing MODIS Active Fire Data", *IEEE Transactions on Geoscience and Remote Sensing*, **Vol. 47**, Nž 1, january (2009).
9. Davis Instruments. "Wireless Vantage Pro2 and Vantage Pro2 Plus Station", DS6152/6162/6153/6163 Rev C, may, (2007).
10. Kazuya Kaku. "Sentinel Asia contributing to Disaster Management Support in the Asia-Pacific Region" Space Applications and Promotion Center, Japan Aerospace Exploration Agency (JAXA) *UNGIWG-8*, Bangkok, Thailand, `http://dmss.tksc.jaxa.jp/sentinel`, november (2007).
11. L. Kooistra, A. Bergsma, B. Chuma and S. de Bruin. "The development of a dynamic web mapping service for vegetation productivity using remote sensing and in situ sensors in a sensor web based approach", Centre for Geo-Information, Wageningen University, *workshop Sensing a Changing World*, november (2008).
12. A. Lopes. "FireStation User's Manual", Universidade de Coimbra, (2000).
13. A. Lopes, M. Cruz and D. Viegas. "FireStation - an integrated software system for the numerical simulation of fire spread on complex topography", *Environmental Modelling and Software*, 17(3):269285, (2002).
14. Arthur Na and Mark Priest. "Sensor Observation Service", *Open Geospatial Consortium Document 06-009r6*, october, (2007).
15. International Association of Oil and Gas Producers (OGP). "OGP Surveying and Positioning Guidance, Note number 7, part 1 - Using the EPSG Geodetic Parameter Dataset", `http://www.epsg.org`, november (2009).
16. A. Pina, B. Oliveira, J. Puga, A. Esteves and A. Proença. "CROSS-Fire: a risk management decision support system on the Grid", *2nd EELA-2 Conference*, Choroní, Venezuela, pp. 25–27, (2009).
17. Peter Schut. "OpenGIS Web Processing Service", *Open Geospatial Consortium Document 05-007r7*, june (2007).
18. Ingo Simonis. "OpenGIS Sensor Planning Service Implementation Specification", *Open Geospatial Consortium Document 07-014r3*, august (2007).
19. I. Simonis, A. Wytzisk and J. Echterhoff. "Sensor Web Enablement: The 52North Suite", *Proceedings of the Free And Open Source Software for Geoinformatics (FOSS4G)*, Lausanne, Switzerland, pp. 11–15, (2006).
20. United States Geological Survey (USGS). "MODIS Overview", `https://lpdaac.usgs.gov/lpdaac/products/modis_overview`, october (2008).
21. A. Whiteside and J. Evans. "Web Coverage Service (WCS) Implementation Standard", *Open Geospatial Consortium Document 07-067r5*, march (2008).
22. N. Yamamoto, R. Nakamura, H. Yamamoto, S. Tsuchida, I. Kojima, Y. Tanaka and S. Sekiguchi. "GEO Grid: Grid Infrastructure for Integration of Huge Satellite Imagery and Geoscience Data Sets", *Proceedings of the 6th IEEE/ACM International Conference on Computer and Information Technology (CIT)*, (2006).