istSOS, a new sensor observation management system: software architecture and a real-case application for flood protection

M. CANNATA*, M. ANTONOVIC, M. MOLINARI and M. POZZONI
Institute of Earth Sciences, University of Applied Sciences and Arts of Southern Switzerland (SUPSI), Campus Trevano, CH-6952 Canobbio, Switzerland

(Received 29 April 2013; accepted 1 November 2013)

istSOS (Istituto scienze della Terra Sensor Observation Service) is an implementation of the Sensor Observation Service (SOS) standard from the Open Geospatial Consortium. The development of istSOS started in 2009 in order to provide a simple implementation of the SOS for the management, provision and integration of hydro-meteorological data collected in Canton Ticino (Southern Switzerland). istSOS is an Open Source, entirely written in Python and based on reliable software like PostgreSQL/PostGIS and Apache/mod_wsgi. This paper illustrates the latest software enhancements, including a RESTful Web service and a Web-based graphical user interface, which enable a better and simplified interaction with measurements and SOS service settings. The robustness of the implemented solution has been validated in a real-case application: the Verbano Lake Early Warning System. In this application, near real-time data have to be exchanged by interregional partners and used in a hydrological model for lake level forecasting and flooding hazard assessment. This system is linked with a dedicated geoportal used by the civil protection for the management, alert and protection of the population and the assets of the Locarno area. Practical considerations, technical issues and foreseen improvements are presented and discussed.

Introduction

In recent years, there has been a tangible effort from the political, scientific and technological domains to establish data interoperability. These actions stem from the commonly accepted assumption that interoperability allows to integrate information coming from different sources and reuse data collected with different purposes. The rationale is that interoperability leads to more accurate and comprehensive information, eliminates duplication and harmonization costs, and thus allows better understanding and ultimately wiser decision-making.

In the geospatial sector, this vision resulted in the establishment of a series of legislative instruments at different levels that encourage data producers to follow a series of standards. Examples include the INSPIRE Directive (Directive 2007/2/EC) at the European level and the Swiss Federal Act on Geoinformation (Geoinformation Act, GeoIG) (RS 510.62) at the national level. Following these frameworks, the scientific community started to develop a large number of research projects that studied new solutions, tested proposed standards and developed new software such as GeoNetwork, GeoServer, pycsw and ZOO (Bleier et al. 2009; ORCHESTRA 2010; TRIDEC 2013).

*Corresponding author. Email: massimiliano.cannata@supsi.ch

© 2013 Taylor & Francis
So far, most of these actions (research and legislations) have focused on leveraging typical Geographical Information System (GIS) layers, such as land use, water protection zones, risk areas, elevation models, hydrological network, etc. As a result, several Web services offering these types of data following defined standards are currently implemented and used in productive environments. Successfully applied standards include Web Map Service (WMS) (OGC 2006) to access maps as images; Web Feature service (OGC 2010a) to transmit geospatial vector data; Web Coverage Service (OGC 2010b) to transmit distributed geospatial data and Catalogue Service for the Web (OGC 2007b) to discover geospatial data and relative metadata on the Web.

In the context of risk analysis (and early warning systems), in addition to the base and contextual data, historical and real-time records of environmental variables are crucial to correctly model, estimate and forecast impending hazards and their expected impacts. Therefore, the accessibility for monitoring data without the need of tedious, error-prone and costly operations of data integration is certainly a goal to be pursued. In the early 2000s Delin and Jackson (2001) and Delin (2002) have proposed the “sensor Web” that aims at developing a standard integration layer capable to harmonize sensor protocols and interfaces in order to drastically reduce the high and extensive effort of manual bridging between sensor resources and applications. In this context, a series of initiatives like the Global Earth Observing System of Systems (Zyl et al. 2009), the intergovernmental Group on Earth Observations, the European Earth observation programme Global Monitoring for Environment and Security (Brachet 2004) and the Open Geospatial Consortium (OGC) Sensor Web Enablement Working Group (Botts et al. 2008) have been launched. The results of these initiatives include case studies, information technology (IT) architecture best practices, Web services and definition of new standards, among others.

The risk management system for the Verbano Lake flooding

The Institute of Earth Sciences (Istituto scienze della Terra, IST) has managed the hydro-meteorological monitoring network of Canton Ticino, Switzerland, since the end of the 1970s. Its hydrometric stations complement a federal network of major rivers, while its rain-gauge stations integrate the MeteoSwiss meteorological monitoring network. During these years, the data quality has been assured through a mixed process of automatic and manual data checks and frequent inspection of the sensors. Since 1979, this annual work of data collection and validation has been leading to the redaction and publication of the “Hydrological Yearbook of Canton Ticino”, edited by the IST, which includes statistics and comments about the hydrological year.

In addition to being used for scientific analyses and general land-management activities, the data collected are actively used to support the assessment and management of the risk of flooding in the Locarno area. In fact, the watersides of the Verbano Lake are particularly prone to flood hazard due to the large catchment size (6599 km²) relative to the small lake surface (212 km²) and limited outlet capacity. With a return period of less than 10 years, the area faces flood events, which usually occur in the fall and are severe in magnitude.

For this reason, in 1997, after yet another severe flood event, the IST, MeteoSwiss, the Locarno e Vallemaggia civil protection agency, the Canton Ticino (TI) Land Department and other cantonal authorities established a specific working group for risk reduction. To this end, the IST, between 1999 and 2002 set up an Early Warning
System for the Verbano Lake which collects data, forecasts hydrological conditions and dispatches alerts to local stakeholders.

**Technological change**

In recent years, due to the evolution of technology, the hydro-meteorological monitoring network of the TI underwent a complete renewal, with new instruments and new data transmission protocols. Sensors changed from being passive devices, capable of providing data only on demand (analogic modems), to being active instruments (GPRS modems), always connected to the Web and capable of pushing information in real time. This hardware upgrade raised the need to renew also the IT part of the system, that includes data storage, data quality assessment and validation and data processing for model input generation.

In the pursuit of data interoperability and IT system redesign, in 2009 the IST became interested in the Sensor Observation Service (SOS) standard (OGC 2007a). However, the requirements identified by experienced IST hydrologists, such as the support of irregular time series, robust quality control mechanism, data aggregation capability and observation correction feature, were not available on existing SOS products (Cannata & Antonovic 2010). As a result, the IST decided to develop its own software implementation of the standard that includes specific extending capabilities: istSOS. In 2010, the software became operational, while in the subsequent years, due to a series of further identified needs, istSOS evolved from a geoservice to a management system.

In the next sections, this paper presents the newly developed “sensor observation management system” based on the SOS standard. After the description of the software components and architecture, the paper illustrates its application in a real-case study. Finally, some conclusions are summarized.

**istSOS: a new sensor observation management system**

istSOS (http://istgeo.ist.supsi.ch/site/projects/istsos) is a sensor observation management system that allows to manage and dispatch observations collected from monitoring sensors. The system follows an interoperable approach, thanks to the adoption of the SOS version 1.0 specifications (OGC 2007a), and is distributed as Open Source Software under the general public license (GPL).

**SOS standard version 1.0.0**

The SOS standard is based on five key elements illustrated in figure 1. The Observations are the data that have been recorded: each measurement is related to a single Procedure that is the sensor, device or process that produced the observations; in the majority of cases the Procedure is the instrument (i.e. rain gauge), while the Observations are the measurements (i.e. millimetres of water). Each Observation is a quantification of one observedProperty, which identifies the target phenomenon (i.e. the rainfall). Moreover, each Observation is a quantification of a phenomenon related to one or more featureOfInterest, which indicates the sampled location(s) or object(s) (i.e. the position of the instrument). A number of procedures are then associated with one or more offerings, which represent a logical group of sensors (i.e. Ticino’s meteorological network) used for easing data access.
The elements in figure 1 are made accessible by SOS through a number of supported requests that are grouped in three profiles: core, transactional and enhanced. The requests can be invoked using Hypertext Transfer Protocol (HTTP) POST or GET protocols and return a standard response in Extensible Markup Language (XML) format.

The core profile, which is mandatory, includes the GetCapabilities, DescribeSensor and GetObservation requests. GetCapabilities allows to discover the specific service metadata (provider, contact, accessibility, etc.) and the offered Procedures, Offerings, Observations and FeatureOfInterest. DescribeSensor is invoked to retrieve detailed metadata of the Procedure, which may include sensor type, producer, owner and technical specifications. GetObservation is the key request for accessing Observations: this operation permits to filter the observed measures by time (intervals or instants), observed properties, spatial conditions or offering.

The transactional profile, which is optional, enables the capability of feeding the system new information using a standardized approach by means of the RegisterSensor and InsertObservation operations. The RegisterSensor permits to remotely register a Procedure specifying: an observation template that describes its Observations detailing the ObservedProperties and FeatureOfInterest, and a description that details metadata in a supported XML format. The sensors are generally described with the Sensor Model Language (SensorML) that is a data model and XML encoding for describing sensor metadata. As a response, the client receives a unique Procedure identifier to be used in future InsertObservation requests whose purpose is to insert new Observations that apply to that specific Procedure.

The enhanced profile is optional. It groups a number of requests whose aim is to facilitate the direct access to specific information. For example, the GetObservationById operation permits to retrieve a specific Observation, whereas the GetFeatureOfInterest allows to retrieve a Geography Markup Language (GML) feature of a specific FeatureOfInterest.
**Software architecture and components**

The istSOS software is composed of two different services and a Web-based interface: the *istsos* service exposes information according to the SOS standard, the *wa* service offers administration management features and the *wainterface* interface allows easy interaction with the two services.

The server side of the application (*istsos* and *wa* services) is entirely written in *Python* (Van Rossum 2003) and relies on the *Apache HTTP server* (Apache Software Foundation 2013) enhanced by the module *mod_wsgi* (Dumpleton 2013), which allows hosting of any Python application supporting the Python Web Server Gateway Interface (WSGI) interface (Phillip 2011). Moreover, it uses *PostgreSQL* (Riggs & Krosing 2010; Worsley & Drake 2002) with the *PostGIS* (Obe & Hsu 2011) spatial extension as the database for data storage, access and elaboration. Other Python packages used by istSOS are *isodate* (http://pypi.python.org/pypi/isodate/0.4.0) for ISO8601 standard conversions, *psycopg2* (http://packages.python.org/psycopg2) for database connections, *pytz* (http://pytz.sourceforge.net) for time zone management and *requests* (http://docs.python-requests.org) for HTTP interaction with other services.

The client side of the application is completely written in *Javascript* and takes advantage of the *ExtJS* (Sencha 2013; Kumar 2012) library, the *dygraph* (Vanderkam 2008) library and *CodeMirror 2* (http://codemirror.net).

istSOS has a nested architecture (see figure 2) where *istsoslib* is the kernel that enables SOS provision, *walib* is the pulp that implements the classes and functions required by the *wa* service and, finally, *wainterface* is the peel that wraps the other components in a user-friendly interface.

The next sections detail technologies, architectures and features for each of the main components.

**istsoslib.** *istsoslib* is the core part of the system that enables the SOS standard provision through the *istsos* service. It has been developed following an abstract factory-pattern approach that allows the initialization of specialized classes or methods built to execute a particular required task. This method provides an interface for creating

![Figure 2. istSOS components and their interactions.](image-url)
families of related or dependent objects without specifying their concrete classes: in this way the application does not even know what class it is going to use, it just gets the factory and then calls a factory method. The library implements three modules: filter, responder and renderer. When a request is submitted to the service, the modules are sequentially instantiated in this order: filter, responder and renderer.

Each module, when instantiated, calls the Factory method that, depending on the request, initializes the specialized object able to execute the desired task. The filter is responsible for verifying and converting the HTTP request (GET or POST) into a Python object, the responder takes charge of the processing operations – information gathering and elaboration or transactional operations – and finally the renderer converts the request results into a valid SOS response.

In addition to standard functionalities, as defined in the SOS documentation, istsoslib implements a series of extending features designed to support the IST data management needs. These features are as follows:

1. **Virtual procedures**: ability to offer data that are not actually measured but are calculated from data belonging to other procedures. A typical example is the provision of river discharges that are computed from water-depth observations using a rating curve.

2. **Irregular time series**: event-driven observation can provide no data in the absence of the event (e.g. no measures without rain), this should be distinguished from missing data (missing observation during event occurrence).

3. **On-the-fly aggregation**: user can request aggregated data of observed properties by specifying an aggregation time resolution, an aggregation mode (maximum, minimum, average or sum) and a no-data value to represent intervals with no observations.

4. **Different output formats**: when observations are requested, the user can specify the desired output format; aside the XML format, istsos supports application/json and plain/text formats.

5. **Insert many at once**: the specifications do not clearly define if with a single insertObservation (note the singular) more than one observation can be registered: istsos supports it.

6. **Observation overriding**: sometimes data managers need to delete or replace existing observations. A typical example is the deletion of test data recorded during instrument verification (e.g. emptying a bucket of water in a rain gauge to verify the correctness of the registered water volume). Another typical example is the updating of observations with new records: when data are collected from other sources, they are often provided at regular intervals with overlapping periods.

7. **Maximum data period per request**: the system allows to set a maximum time period for a single request when executing a getObservation; this prevents server overload and service unavailability.

8. **LIKE wise support in observed properties filtering**: observed properties are generally organized in a hierarchic relation each other according to a well-defined ontology. For example, temperature and humidity are both meteorological properties of the air, so they are likely registered as urn:ogc:def:property:x-istsos:1.0: meteorological:air:temperature and urn:ogc:def: property:x-issost:1.0: meteorological:air:humidity. With the LIKE wise filter capability, user can request for air-observed property and both are returned.
(9) **Data quality index**: istSOS supports natively the data-quality index as an indispensable property of the data. In fact, data have no, or very poor, value if no information regarding their quality is provided. Each registered observation and each position of any mobile instruments, is associated with a data-quality index. By default, istSOS provides seven quality level codes: raw, erroneous, reasonable, statistically sound, timely coherent, spatially coherent, manually adjusted, correct. The levels are assigned sequentially at the passing of quality-check tests. The user has the ability to select a constraint for an ObservedProperty (e.g. rainfall $> = 0$) and a constraint for each combination of Observed-Property Procedure (e.g. $-10 < \text{temperature in Lugano} < +35$); if set, these are used to respectively assign the erroneous, reasonable and statistically sound quality codes directly during data insertion. In addition, users have the ability to define their own quality index levels, update the Observations quality indexes with an insertObservation request and retrieve observations using the quality index to filter the data.

**IstSosLib** allows to define multiple istSOS service instances, each one with its specific configuration that extends a system default configuration. Each service is identified by a unique name, which is also the name of the database schema where data are stored. In **Figure 3**, the entity-relationship model (ER model) of a single istSOS instance is represented. The registered Procedures are stored in the procedure table that is connected with the foi table (FeatureOfInterest) and the offering through a cross-reference table (offering_has_procedures). Any Observation is stored in the measures table that is linked to a specific time instant (event_time table) element, a specific data-quality code (quality_index table) element and a Procedure-Observed-Properties and UnitOfMeasure triplet element (linked together in the proc_obs table). A constraint guarantees that for a single time instant and one single proc_obs element, there is only a single measured value.

For further details on the istsos component, interested readers can refer to Cannata and Antonovic (2010) and to istSOS online documentation.

![Figure 3. istSOS database schema.](image-url)
walib. walib is the pulp of the system that primarily enables the wa service and, through it, the setting and configuration of istSOS using a “REpresentational State Transfer” (RESTful) (Richardson & Ruby 2007) approach. This newly developed component, in accordance with the RESTful paradigm, allows to access the elements by using the HTTP methods: GET to retrieve, POST to add, PUT to update and DELETE to remove objects.

Figure 4 shows the exposed resources identified in a directory-structure-like uniform resource identifiers (URIs) with information on the supported methods.

walib, adopts the json format (http://www.json.org) as the communication format between the client and the server. In general, the GET method just requires to specify the target element in the URI; the POST requires to provide the same parameters obtained with a GET operation on the same element with the exclusion of the object identifier; the PUT has to send all the parameters obtained with a GET operation on the same element, including the object identifier and, finally, the DELETE requires to provide the element identifier specified in the URI.

A generic wa response includes the following members: “success” which indicates if the request was successfully executed, “message” which defines a returned textual note (particularly useful if errors occur), “total” which identifies the number of returned elements and “data” which lists the elements with their attributes and structure. Note that “total” and “data” are returned only if data are actually requested.
To better clarify how to interact with the wa service, in the following paragraphs a series of examples of communications with a service registered at the URL http://localhost/istsos/wa/ with reference to the data quality element (dataqualities) are presented. Each request will report the used HTTP method, the URI for the request, the Request Body and the Response:

Gather information about the quality index with code 200 for the service demo:

**Method:** GET

**Address:** http://localhost/istsos/wa/istsos/services/demo/dataqualities/200

**Request Body:** {}

**Response:** {“success”: true, “message”: “Data qualities of service <demo> successfully retrieved”, “total”: 1, “data”: [{“code”: 200, “name”: “reasonable”, “description”: “the value is in a reasonable range”}]}

Update the properties of the quality index with code 200:

**Method:** PUT

**Address:** http://localhost/istsos/wa/istsos/services/demo/dataqualities/200

**Request Body:** {“name”: “reasonable”, “description”: “the value is in a statistical range”}

**Response:** {“message”: “Update successful”, “success”: true}

Insert a new quality index:

**Method:** POST

**Address:** http://localhost/istsos/wa/istsos/services/demo/dataqualities

**Request Body:** {“code”: 201, “name”: “correct”, “description”: “the value is correct and manually checked”}

**Response:** {“message”: “New element added”, “success”: true}

Delete the quality index with code 201:

**Method:** DELETE

**Address:** http://localhost/istsos/wa/istsos/services/demo/dataqualities/201

**Request Body:** {}

**Response:** {“message”: “Element successfully deleted”, “success”: true}
In addition to common actions aiming at manipulating elements, the walibs offers features to perform general operations like service-status evaluation or a database-connection check.

**wainterface.** wainterface has been primarily designed to provide a user-friendly environment for setting up and configuring istSOS services. The interface has a menu allowing to access different sections and a tool bar with section-specific buttons activating different frames.

The currently available sections are: server, services and data viewer. When the server section is selected, the interface shows a toolbar with buttons to access the following features:

1. **About:** provides information about the installed version of istSOS, available updates, news and general information about istSOS with relevant links to documentation, mailing list and code repository.
2. **Status:** here the administrator has an overview of instantiated services (see figure 5) including the number of registered FeatureOfInterest, Offerings, Procedures and ObservedProperties. Moreover, it provides the availability of the services, the corresponding database and the basic accessible requests (GetCapabilities, DescribeSensor, GetObservation, GetFeatureOfInterest, InsertObservation and RegisterSensor).
3. **Database:** form for setting and testing the istSOS default connection to a PostgreSQL/PostGIS database.
4. **Service provider:** form for setting the default service provider information, like contacts and address.
5. **Service identification:** form for setting the default service information regarding keywords, access fee and authority.
6. **Coordinate systems:** form for setting the default coordinate system used for storing geospatial geometries, other supported reference systems and urn definition for easting, northing and elevation.
7. **GetObservation configuration:** form for setting the maximum supported hours in a data request period, quality indexes and values to be used in case of no data in aggregation requests and by default.
8. **Proxy configuration:** this is the link to the istSOS service as accessible from the Web. This URL is used by the walib to communicate with the istoslib, so that, for example, when an insert sensor request is submitted with the RESTful interface, the walib code generates a SOS RegisterSensor request in XML and forwards it to the istSOS service at the address here configured.

![Figure 5. Screenshot of the wainterface: Status feature of the Server section.](image)
(9) **New service:** here the administrator accesses a wizard for the creation of new services.

(10) **Delete service:** from this frame the administrator can delete an existing service; note that all the related information like procedures sensorML and observations are permanently deleted.

The Services section allows to select an existing service and update its information (see figure 6). A first set of features reflects those available for the server section, namely database, service provider, service identification, coordinate system and GetObservation configuration. A second set of buttons are as follows:

(1) **Offerings:** here registered procedures can be associated with offerings according to the administrator preferences. Note that offerings, being a mandatory parameter in getObservation request, can be used to permit and restrict access to observations.

(2) **Procedures:** lists registered procedures and provides links for modifying related metadata (sensorML) or deleting those selected.

(3) **New procedure:** offers a form to register a new procedure. Using its inputs, wa generates a RegisterSensor request that is forwarded to the istsos service. Here a check on mandatory parameters is performed.

(4) **Observed Properties:** allows to register observedProperties definitions to be used by new procedures.

(5) **Unit of measures:** permits to register unit of measure definitions to be used by new procedures.

(6) **Data quality:** allows to register qualityIndex definitions.

The third section of the interface is the data viewer (see figure 6). In this interface, a panel named data editor provides access to observations display and editing. Once the desired time series (service, offering and procedure) are selected, the administrator has to define a time interval and an observedProperties: at this point observations are plotted in a chart and displayed in a table. If desired, an editing session can be
started (see figure 7). Within the editing session, the administrator can select observations, either interactively on the plot or on the table. Selected observations can be updated by either editing the table rows or applying formulas, thanks to the calculator which also allows us to refer to data from other series. Modifications are immediately visualized and can be saved by pressing the save button.

A different panel, which is part of the \textit{wainterface} but not integrated in it, is the data display. It is a stand-alone Web page that can be used to represent observations, according to administration set rules and permissions. It is graphically similar to the data editor but lacks editing capabilities.

**Real-case application**

Whenever structural interventions are not feasible, the damage reduction due to flood events relies on a solid early warning system. In fact, timely warning is crucial for efficient emergency response and contingency action planning. For this reason, as stated in the introduction, a system for Early Warning on the Verbano Lake was set up by the IST between 1999 and 2002. Today it is still operational and presents improved capabilities that reflect IT’s advances. The system is currently composed by three main blocks: data acquisition and management, lake level forecasts and intervention support system. These blocks are described in the next sections.

**Hydro-meteorological data acquisition and management**

To provide data for the forecasting model of the Verbano Lake level, the system has to collect information related to the lake catchment: both observed in the field (field true) and forecasted by meteorological models. This catchment is a trans-national basin located across Switzerland and Italy. In Switzerland, it is monitored by different agencies for different purposes: MeteoSwiss for meteorological scope, FOEN (Federal Office of ENvironment) for river conservation and TI for land-use planning. In Italy, it is monitored by Piemonte regional environmental agency (ARPA, Agenzia Regionale per la Protezione dell’Ambiente) and northern Italy lakes consortium (CGL, Consorzio Grandi Laghi). As a result, field observations of
hydro-meteorological data have to be collected in near real time from several agencies and homogenized in order to provide consistent inputs for the forecasting model. Moreover, due to international conventions, data should be exchanged between the partners.

In the past, field instruments of TI and FOEN were equipped with GSM modem, so at defined time intervals (1 day or 2 hours in case of meteorological alert), the IST executed batches that called the sensor and downloaded the data. Then, MeteoSwiss, ARPA and CGL provided ASCII files through File Transfer Protocol (FTP) with updated observations. Transmissions occurred at given intervals and each dataset had overlapping periods between consecutive dispatches but with updated observations.

It is clear that the system is redundant, inconsistent and error prone. It is redundant because each partner has to manage data already managed by other agencies. It is inconsistent because collection of late observations due to temporary instrument unavailability or correction of measures due to instruments deficiencies leads to no unique datasets. It is error prone, because FTP protocol does not provide appropriate transmission-error management and data-integrity-checking mechanism; thus errors cannot be automatically caught and appropriate patching action cannot be automatically triggered.

For all the above reasons, it would be advised that all the partners move towards a service-oriented architecture using a standard protocol. In this vision, the IST decided to test, validate and promote the adoption of one available standard for serving observations.

Today, using istSOS, the IST offers the TI monitoring network data with the SOS standard. Thanks to the GPRS, observations from 136 sensors deployed in the field flow into the system real-time measurements of: rainfall, air temperature, air humidity, atmospheric pressure and solar radiation, river heights, river temperature and river discharges, lake level and lake temperature. More than 35 million measurements since 1978 are currently accessible on demand. Moreover, the IST integrated in this system has approximately 3.5 million measurements a year coming from 111 sensors belonging to partners. Overall, the system records a monthly data flux of about 5 GB. Requesting 1 day of temperature data at 10 minutes resolution from one station takes approximately 250 milliseconds, while with hourly average aggregation, the same request requires about 450 milliseconds.

In addition, high-resolution (about $7 \times 7$ km) meteorological forecasts are received from MeteoSwiss in grib format. These are archived in a hybrid system that uses a database for metadata recording together with geoTIFF files for data serving and retrieval.

**Hydrological model**

The lake level forecast is based on a conceptual hydrological model, where the main parameter (curve number) is calibrated at the beginning of each simulation, while other less important parameters were calibrated using data of historical events. The results of the model are the water discharges in the main tributaries of the Verbano Lake and the lake level. The forecasts have a 3-day horizon and are transmitted through fax, SMS, ftp and e-mail to all the partners. The model has been operative for more than 10 years and has proven several times its consistency, in particular during the 2000 and 2002 flood events in the Verbano Lake.
In the last 3 years, due to a complete change in the database infrastructure, the model underwent various improvements. The model inputs are now coming from a Web Processing Service (OGC 2007c) process that

- collects data from the SOS;
- dynamically computes basins total observed rainfall as a result of data interpolation and areal statistics;
- collects data from weather forecasts;
- dynamically computes basins total forecasted rainfall as a result of areal statistics;
- formats and generates model input files.

The model core remains the same, but the interface was improved with a better visualization of graphics. The model output is transmitted to the partners and to the designated authority’s stakeholder. Moreover, it is the crucial input for the SITGAP system described in the next section.

**SITGAP**

SITGAP is the Italian acronym for: Geographical Information System for the Management, Alarm and Planning of Verbano Lake flooding interventions. The system is basically a Web-mapping interface that allows accessing a series of information and tools supporting the local Civil Protection in mitigation and evacuation action planning. The application allows users to select four different operational levels. Normal level – no floods are expected – allows to administer and retrieve all the emergency institution information, including contact points. Alarm level – floods are likely expected – permits users to query and visualize forecasted water levels and corresponding flooded areas, exposed elements and relative information. Action level – an intervention has been activated – allows users to plan and control field intervention and allocated resources. Evacuation level – severe flood in place – allows users to manage evacuation operations, including registration of evacuated persons with destination and searches by person or address.

The Web portal is currently based on Neapoljs for ESRI ArcIMS and ESRI ArcSDE for Oracle, but a new improved version is under development and will be likely released in 2013.

**Conclusions and further works**

The IST-SUPSI has studied the application of the SOS in a real-world scenario; from this experiment the authors have identified a series of requirements not addressed by the standard. Thus, they have engineered their own software solution which, thanks to specific extending features, fills the identified gaps.

Like other OGC services, the SOS is designed to be machine-readable. In fact, it is difficult to manually handle data coming from this type of services (mostly XML): think of what WMS will be without visualizers like OpenLayers. Similarly, SOS, to be really used by scientists and interested public users, requires graphical interfaces that simplify the human interaction. For this reason, a RESTful service capable of exposing system setting and generating SOS requests was developed. A graphical
user interface was then designed to ease the access to these features. As a result, the user is now able to easily manage the system, create new services, register new sensors and insert observations without any XML exposure.

The system in place for the Verbano Lake flooding provided an ideal real case to verify the robustness of the implemented solution. Thanks to istSOS, the IST’s scientists have now access to a centralized point that aggregates information despite differences in the sensor manufacturer, observation type or measured property. Moreover, hydrologists can now rely on a more flexible and consistent system which, for example, takes automatically advantage of newly available observations for forecasting the lake level or provides alerts whenever inactive or corrupted sensors are identified.

As a final comment, this paper shows that SOS has definitely an enormous potential. Nevertheless to be fully operational, some gaps remain have to be filled and new interfaces are required. Among other features, SOS version 2.0.0 (Bröring et al. 2010) solves some of the version 1.0 issues (e.g. name inconsistencies), introduces new bindings (key-value-pair and SOAP) and redesigns the observation-offering concept. Even if these solutions do not solve most of the issues we have highlighted in this paper, they represent a promising advance in the topic of sensor Web. For this reason, future work will extend istSOS to support the version 2.0.0 of the SOS standard and will introduce two major modifications to faster data access of long measurement series: istSOS database schema will use one measurement table for each Procedure, producing tables with fewer rows, and istSOS service will use the Tornado Web Server (Dory et al. 2012), which is known for its high performance.

Acknowledgements
The authors wish to thank all the partners of the Verbano Lake working group for flood risk reduction, and especially the Canton Ticino (Ufficio dei corsi d’acqua) for supporting this project. Special thanks to Dr Fleming and CSIR for contributing to the software development and Francesco Massa for testing and documentation. Finally, Dr Fabio Lepori and Jeff McKenna are thanked for revising the document.

References
Apache HTTP server version 2.2 documentation [Internet]. c2013. The Apache Software Foundation; [cited 2013 Nov 21]. Available at: http://httpd.apache.org/docs/2.2

Bleier T, Bozic B, Bumerl-Lexa R, Da Costa A, Costes S, Iosifescu I, Martin O, Fryninger S, Havlik D, Hilbring D., et al. 2009. SANY – an open service architecture for sensor networks [Internet]. SANY Consortium; [revised 2009; cited 2013 Nov 21]. http://www.frisia-it.de/assets/images/SANY_Book.pdf

Botts M, Percivall G, Reed C, Davidson J. 2008. OGC sensor web enablement: overview and high level architecture. GeoSensor network. In: Silvia N, Alexandros L, Anthony S, editors. Berlin: Springer-Verlag; p. 175–190. doi:10.1007/978-3-540-79996-2_10.

Brachet G. 2004. From initial ideas to a European plan: GMES as an exemplar of European space strategy. Space Policy. 20:7–15.

Bröring A, Stasch C, Echterhoff J. 2010. OGC Implementation Specification: Sensor Observation Service (SOS) 2.0 Candidate Standard (10-037) [Internet]. Open Geospatial Consortium Inc.; [revised 2012 Apr 20; cited 2013 Nov 21]. Available from: http://www.opengeospatial.org/standards/sos

Cannata M, Antonovic M. 2010. istSOS: investigation of the sensor observation service. In: Brovelli MA, Dragicevic S, Li S, Veenendaal B, editors. ISPRS Archives – Volume
XXXVIII-4/W13. Proceedings of the WebMGS 2010: 1st International Workshop on Pervasive Web Mapping, Geoprocessing and Services; 2010 Aug 26–27; Como: Politecnico di Milano.

Delin KA. 2002. The sensor web: a macro-instrument for coordinated sensing. Sensors. 2:270–285.

Delin KA, Jackson SP. 2001. Sensor web: a new instrument concept. In: Descour MR, Rantala JT, editors. Functional Integration of Opto-Electro-Mechanical Devices and Systems. Proceedings of SPIE’s Symposium on Integrated Optics; 2001 Jan 20–26; San Jose, CA.

Dory M, Parrish A, Berg B. 2012. Introduction to Tornado. Sebastopol (CA): O’Reilly Media.

dygraphs JavaScript visualization library. [Internet]. c2008. Vanderkam D.; [cited 2013 Nov 21]. Available from: http://dygraphs.com

Ext JS Sencha docs [Internet]. c2013. Sencha; [cited 2013 Nov 21]. Available from: http://docs.sencha.com

Kumar A. 2012. Sencha MVC architecture. Birmingham: Packt Publishing Ltd.

Mod_wsgi documentation [Internet]. c2011. Graham Dumpleton; [cited 2013 Nov 21]. Available from: https://modwsgi.readthedocs.org modwsgi

Obe R, Hsu L. 2011. PostGIS in action. Stamford: Manning Publications.

Open Geospatial Consortium. 2006. OpenGIS® web map server implementation specification [Internet]. Open Geospatial Consortium Inc.; [revised 2006 Mar 15; cited 2013 Nov 21]. Available from: http://www.opengeospatial.org/standards/wms

Open Geospatial Consortium. 2007a. OpenGIS® sensor observation service. OpenGIS® Implementation Standard. [Internet]. Open Geospatial Consortium Inc.; [revised 2007 Oct 26; cited 2013 Nov 21]. Available from: http://www.ogcnetwork.net/SOS

Open Geospatial Consortium. 2007b. OpenGIS® catalogue services specification [Internet]. Open Geospatial Consortium Inc.; [revised 2007 Feb 23; cited 2013 Nov 21]. Available from: http://www.opengeospatial.org/standards/specifications/catalog

Open Geospatial Consortium. 2007c. OpenGIS® web processing service [Internet]. Open Geospatial Consortium Inc.; [revised 2007 Jun 8; cited 2013 Nov 21]. Available from: http://portal.opengeospatial.org/files/?artifact_id=24151

Open Geospatial Consortium. 2010a. OpenGIS® web feature service 2.0 Interface Standard (also ISO 19142) [Internet]. Open Geospatial Consortium Inc.; [revised 2010 Nov 2; cited 2013 Nov 21]. Available from: http://www.opengeospatial.org/standards/wfs

Open Geospatial Consortium. 2010b. OGC WCS 2.0 interface standard – core [Internet]. Open Geospatial Consortium Inc.; [revised 2010 Oct 27; cited 2013 Nov 21]. Available from: http://www.opengeospatial.org/standards/wcs

ORCHESTRA Project portal [Internet]. c2010. ORCHESTRA Consortium; [cited 2013 Aug 13]. Available from: http://www.eu-orchestra.org [Internet].

Phillip JE. 2011. Python web server Gateway Interface v1.0 [Internet]. c1990–2013. Python Software Foundation; [cited 2013 Nov 21]. Available from: http://www.python.org/dev/peps/pep-0333

Richardson L, Ruby S. 2007. RESTful web services. 1st ed. Sebastopol: O’Reilly Media.

Riggs S, Krosing H. 2010. PostgreSQL 9 administration cookbook. Birmingham: Packt Publishing.

TRIDEC Project portal [Internet]. c2013. The TRIDEC consortium; [cited 2013 Nov 21]. Available from: http://www.tridec-online.eu

Van Rossum G. 2003. The Python language reference manual. Bristol: Network Theory Ltd.

Worsley J, Drake JD. 2002. Practical PostgreSQL: a hardened, robust, open source database. Sebastopol (CA): O’Reilly Media.

Zyl TLV, Simonis I, McFerren G. 2009. The sensor web: systems of sensor systems. Int J Digital Earth. 2:16–30. doi:10.1080/17538940802439549