2012 IEEE Eighth World Congress on Services

RESTful Sensor Web Enablement Services for Wireless Sensor Networks

Mohsen Rouached
College of Computers and Information Technology
Taif University
Taif, Saudi Arabia
m.rouached@tu.edu.sa

Sana Baccar, Mohamed Abid
CES Research Unit
National school of Engineers of Sfax
Sfax, Tunisia
{*sana.baccar,mohamed.abid*}*@ceslab.org*

***Abstract*—Due to the large number of sensor manufacturers and differing accompanying protocols, integrating diverse sensors into observation systems is not straightforward. A coherent infrastructure is needed to treat sensors in an interoperable, platform-independent and uniform way. The concept of the Sensor Web reflects such a kind of infrastructure for automatically discovering and accessing appropriate information from heterogeneous sensor devices over the Internet. In this context, the Open Geospatial Consortium (OGC) established the Sensor Web Enablement (SWE) initiative that specifies standard interfaces and encodings to remotely access, encode and exchange the sensed data. However, SWE standards have several gaps that limitate its capabilities to achieve the sensor Web desires. In this paper, we address the problems related to the data format and the architectural style followed by the implementation of the SWE services. Indeed, we propose the adoption of the lightweight Representational State Transfer (REST) web services concept and the usage of JavaScript Object Notation (JSON) format as an alternative to the verbose XML one for the exchanged messages.**

***Keywords*-Sensor Web, Wireless Sensor Networks, Sensor Web Enablement, Representational State Transfer**

I. Introduction

Today, the notion of the sensor web has been largely influenced by the concept of the Internet-Of-Things. It is considered as an infrastructure that enables to collect, model, store, retrieve, share, manipulate, analyze and visualize sensor data/metadata via the World Wide Web (WWW) in a standardized way. Moreover, it acts as an extensive monitoring and sensing system that provides timely and continuous observations. Thus, *the Sensor Web is to sensor resources what the WWW is to general information sources, an infrastructure allowing users to easily share their sensor resources in a well-defined way* [1]. This new earth-observation system opens up a new horizon road to fast assimilation of data from heterogeneous sensors and to provide new knowledge that affects new decisions in the future.

A key challenge in building the Sensor Web is how to automatically access and integrate different types of spatiotemporal data that is observed by heterogeneous sensor devices or generated using simulation models. Another challenge issued by the fact that most applications are still integrating sensor resources through suitable mechanisms, instead of building upon a well-defined and established integration layer.

On the other hand, the sensor networks are currently developed around different communities of sensor and user types, each community typically relying on its own system, metadata semantics, data format and software. Thus, integrating diverse sensors into observation systems is not straightforward. It is typically hindered by incompatible services and encodings which can cause interoperability between different sensor nodes within the same WSN. This issue has been the driving force for the Open Geospatial Consortium (OGC) [1] to start the Sensor Web Enablement (SWE) initiative [2] whose architecture was designed to enable the creation of web-accessible sensor assets through common interfaces and encodings.

Although this intiative allows for fusing multiple data models and formats into a common data model and representation, it has the limitation that it only provides rudimentary support for the required data conversion. In addition, its framework presents some major gaps when it comes to data format and messages exchange, which are XML based. In fact, this format is verbose and therefore is not suitable for low power and data rate devices.

Apart from the data format, the architectural style followed by the implementation of the SWE services can generate serious problems. Indeed, There are several ways to realize the web service concept: some realizations are built on mechanisms that require significant processing power and communication bandwidth, whereas others are more lightweight. The resource constraints inherent in sensors regarding processing power, energy, and communication bandwidth necessitate the use of lightweight mechanisms.

To address these two challenges, we propose a lightweight RESTful approach based on SWE services for interacting with the constrained WSNs. Our proposed approach is based mainly on considering each sensor node as RESTful resources [2] that can be accessed and polled over the SWE services using the lightwight JavaScript Object Notation (JSON) data format as an alternative to the verbose XML

[1]http://www.opengeospatial.org/.
[2] http://www.ogcnetwork.net/swe.

one for exchanging messages [3].

The remainder of the paper is structured as follows: Section 2 depicts the most important related works that deal with the Sensor Web concept and the integration of the service oriented architecture in WSN environments. In section 3, we present our approach that adapts the SWE services to the REST architectural style. Section 4 is dedicated to the experimentation and the performance evaluation of the proposed approach. Finally, Section 5 concludes the paper and outlines some open challenges and future work for SWE.

II. Related work

Several efforts have been invested in order to handle the challenges related to the integration of diverse sensors into observation systems in an interoperable and uniform way and providing a sensor web services that enable web users to efficiency connect and share the heterogeneous sensor resources. The Sensor Web Enablement (SWE) initiative, defined by the OGC, is one among these important efforts. SWE was mainly designed to enable the creation of web-accessible sensor assets by fusing multiple data models and formats into a common interfaces and encodings.

These services and encodings cooperate to create an efficient integration platform that enables users to remotely register, discover, retrieve and ask for observations of sensor resources. SWE services aim also to standardize the vision of Sensor Web but their implementation is still required. Therefore, various middleware technologies have been raised to manage sensor resources and make them available to applications via SWE standards. Among these middleware systems there is the 52 ° North Sensor Web framework [3]. This latter provides implementations to integrate sensor resources with the SWE services through an intermediary layer called the Sensor Bus. Other projects, based on combining SWE initiative with various Web technologies, such as Sensor Web 2.0 [4], have been developed to integrate aspects of the Web-Of-Things with SWE.

On the other hand, no-standardized approaches for building a Sensor Web have been emerged like Hourglass [5], the Global Sensor Network (GSN) [6], the Sensor Network Services Platform (SNSP)[7] and SOCRADES [8]. Although these approaches define their own proprietary interfaces and encodings for data querying, locating and timing, they still need to standardize their services as well as to provide service interfaces for tasking of sensors and pushing sensor data to subscribed users.

Due to its important role in promoting Sensor Web, many research groups have worked to improve the OGC' s SWE specifications by developping services that enable to register and publish the sensor Web achievements. In this scope, the GinisSense-system developers tried in [9] to provide services implementations that improve monitoring and notifying users about sensed observations. For that, they proposed a five-tiers architecture that is based on the OGC' s SWE specifications and the XML format for the exchanged messages. In fact, these implementations are focused on integrating only the SAS and WNS services, implementing the other services like SOS and SPS is still an open issue. Although this system satisfies required communication between end-users and sensor networks, the proposed operations were not implemented. Moreover, there is a shortfall when it comes to processing sensed data and extracting knowledge from it.

In the same context, X.Chu and al. [10] have implemented an Open Source Web Architecture (OSWA). OSWA is a service oriented Sensor Web framework that extends SWE and provides additional services to enable users to remotely process the information collected from ubiquitous sensors via the Internet. After testing the performance and the scalability of the provided services, the researchers identified that the poverty in performance of the query based observation is due to the execution mechanism of TinyDB.

Although this mechanism is a good way to enhance the performance of the query based observations, it is quite hard to determine the period of time for the cache to be valid. Furthermore, T.Kobialka et al. [11] have identified that the more physical access to individual sensors is required, the more it is difficult to update the code on huge number of nodes. To do this, they proposed to deploy a sensor node middleware onto the sensors which acts as an interface to facilitate migration and provide code management.

The shortcomings of this approach are issued from the fact that this middleware is not able to automatically modify the shared interfaces for each operation. In addition to the inability of sensor network to handle more than one query at a time without some special operators or middleware deployed onto the sensors. To overcome this shortfall, they introduced a Cache Manager into the Sensor Observations Service (SOS) architecture which retrieves the asked observations from the sensor networks and checks the transmission of the response to the requester. Nevertheless, other open issues are still in need to be resolved such as that of minimizing the complexity of underlying machine, learning algorithms, encapsulate functions and ensuring interoperability.

Other proposed approaches are based mainly on directly implementing a lightweight Representational State Transfer (RESTful) API on each sensor node in order to guarantee an easy communication between the sensors and ensure an automatic discovery of sensor node as well as an efficient communication with Web-users. [12][13] addressed the feasibility of using RESTful Web services with IP-based WSNs. These works adopted the "IP-based sensor network" System to ensure a smooth communication between nodes over Web services and make the direct integration in modern IT systems easier.

In the same scope, to interconnect different sensor and

[3]http://52north.org/.

actuator nodes in monitoring systems, L.schor et al.[14] identified a Plug&Play approach based on RESTful Application Programming Interface (API). This API is directly implemented on Web Enabled Devices (without using of gateway) in order to automatically discover the sensor nodes in a wireless network. To guarantee that the server has up-to-date sensor information, the sensor node has to to send a periodic message that includes its status. As an alternative, the authors proposed an approach that consists in polling the state of the sensors per each demand of server. This alternative was considered more efficient due to the fact that the states of both the sensors and the actuators can be extracted and modified by the same interface. In spite of its utilities, other needs are still required such as defining and updating the mediator, finding the right services and resolving data mismatching.

Other sensor Web researchers are interested in providing an efficient and rigorous way to retrieve, organize and save heterogeneous sensor data/metadata. Researchers in [15] proposed a new sensor database named RESTful Environmental DataBase (REnvDB) that is useful to interconnect different types of WSNs, extract sensor observations and expose them over the Web. This database provides a common lightweight RESTful interface that grants the access to the collected observations using HTTP operation calls into the URLs and translates exchanged messages through a proper gateway on the WSN' s side. In fact, this approach can reduce communication complexity due to its lightweight connection based on RESTful interface. Moreover, the time that can be reduced on the database side, will be on the other hand increased due to using gateways to forward the sensor observations into the REnvDB. To overcome this shortfall, defining a standardized messages can make gateways dispensed. More immportant related works are exposed in [16] and [17].

After studying the main efforts that aimed to ensure an efficient Sensor Web concept, we identified that using REST architectural style for SWE initiative can improve an efficient architecture that overcomes high consumptions and limited resources constraints. Moreover, to meet the existing SWE services expectations especially concerning their exchanged messages format, we proposed to use the lightweight JSON format.

III. RESTful architecture for SWE

In this section we start by illustrating the system architecture and introducing the main ingredients of our approach based on REST technology and SWE standards. Then, the adaptation features of the REST architectural style for the SWE services are outlined.

A. Representational State Transfer (REST)

REST [2], [18] is an architectural model for how distributed applications are built. Systems built around the REST architecture are said to be RESTful. REST builds on three concepts: representation, state, and transfer:

- Representation: Data or resources are encoded as representations of the data or the resource. These representations are transferred between clients and servers. One example of a representation of a resource is a temperature value written as a decimal number, where the representation is the decimal number and the temperature is the resource.
- State: All of the necessary state needed to complete a request must be provided with the request. The clients and servers are inherently stateless. A client cannot rely on any state to be stored in the server, and the server cannot rely on any state stored in the client. This does not, however, pertain to the data stored by servers or clients, only to the connection state needed to complete transactions.
- Transfer: The representations and the state can be transferred between client and servers.

REST is an architectural model that can be efficiently implemented as a combination of the Hypertext Transfer Protocol (HTTP) and TCP/IP. With this instantiation of REST, HTTP requests are used to transfer representations of resources between clients and servers. Uniform Resource Identifiers (URIs) are used to encode transaction states.

To illustrate a REST transaction, let us consider a smart object system implemented with Web services. Temperature sensors post temperature data to a building automation server. The building automation server configures a radiator based on the temperature data. In this example, the temperature sensors submit their temperature data to the building automation server using the HTTP PUT method. To query sensors, the server uses the HTTP GET method. The server then sends its configuration request to the radiator using the HTTP PUT method. We focus on the server's request for temperature data from one of the sensors. This request is implemented by using the HTTP GET request, which is issued by the server to one of the sensors. The sensor responds with the temperature data of the sensor in JSON format. The HTTP GET request sent by the server is shown as follows

```
GET /sensors/temperature HTTP/1.1
Content-type: application/json
```

The HTTP request, which is human-readable, consists of two lines of text. The first line contains the HTTP GET verb, followed by the URI that represents the temperature sensor. In this case this is as simple as /sensors/temperature, but more complex URIs are possible. Ending the first line is the name and version of the HTTP protocol. The second line of the server's request contains the requested representation of the data that the client has to offer. This line contains the HTTP header *Content-type* followed by the type *application/json* . This type is defined in the JSON specification as the content type to be used for JSON data. The server's response looks as follows:

```
HTTP/1.1 200 OK
Content-type: application/json

{"sensors":[{"name": "Temperature",
"value": 26.1}]}
```

The reply consists of two parts, the HTTP header and the HTTP body. The header is two lines long. The first line contains the HTTP/1.1 keyword, which again tells the receiver that this reply is in HTTP version 1.1 format. This keyword is followed by the HTTP status code 200, which tells the receiver that the HTTP request was successfully processed. The *OK* following the status code is a human-readable representation of the status code. The HTTP reply contains the same *Content-type* header as the request, which tells the receiver that the data in the HTTP body are in JSON format. Following the HTTP header is a blank line that divides the header from the body. The HTTP body contains the JSON data that represent the current temperature as sensed by the smart object's sensor.

B. Sensor Web Enablement (SWE)

The SWE framework consists of a set of standards that define data formats for sensor data and metadata and web service interfaces for providing sensor related functionality. As depicted in Figure 1, the SWE framework can be divided into two parts: the *interface model* defining the interfaces of sensor related web service types and the *information model* comprising those standards which address the specification of data formats.

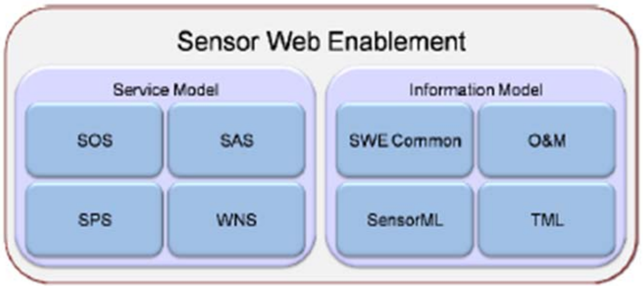


Figure 1. SWE Framwork

1) SWE Information model:

The SWE information model comprises a set of standards which define data models primarily for the encoding of sensor observations as well as sensor metadata. For this purpose, the first generation of SWE contained three specifications: Observations & Measurements (O&M), the Sensor Model Language (SensorML), and the Transducer Markup Language (TML), which define XML shemas for encoding sensor observations and measurements, describing sensors and processes they can participate in, and describing transducers respectively.

TML supports the encoding of sensor data as well as metadata by focusing on data streaming. TML has only been rarely used in practice and has not been further evolved so far. In the new generation of SWE specifications, TML is not referenced anymore and recent conversations in OGC's SWE working group showed that there is no urgent demand in TML and a retirement of the standard is in discussion [1]. Hence, as in [1], we do not see TML as part of the new generation SWE. The Observations & Measurements standard defines a domain independent, conceptual model for the representation of (spatiotemporal) measurement data. It comprises an implementation of this conceptual model as an XML based GML application schema. The Sensor Model Language specifies a model and XML encoding for the description of all kinds of sensor related processes.

2) SWE Interface Model:

The SWE interface model comprises standards that specify the interfaces of the different Sensor web services [1]. Four service interfaces were defined for the first generation of SWE: The Sensor Observation Service (SOS) offers pull-based access to sensor measurements as well as metadata. The Sensor Alert Service (SAS) allows subscribing to alerts in case of a sensor measurement event that fulfills certain criteria. The Sensor Planning Service (SPS) can be used for tasking sensors and setting their parameters. The Web Notification Service (WNS) is, unlike the other three services, not directly sensor related. It is a supportive service which provides asynchronous notification mechanisms between SWE services and clients or other SWE services (e.g., delivery of notifications) including protocol transducing capabilities.

The service specifications of the first generation SWE primarily concentrated on the definition of an XML schema which reflected the service functionality. In the new generation SWE, first a conceptual service model is defined (using UML notation), before an XML implementation of that model is specified. This facilitates the adoption of other forms of implementations of the conceptual model such as the JSON implementation.

C. Adaptation of the SWE framework to the REST architectural style

To enable the necessary interoperability between heterogeneous nodes and offer an accessible services to end users, we propose to adapt the SWE services to the RESTful architecture as shown in Figure 2. This adoption is based on two features. The first one concerns the interface model. The second one is related to the information model and more specifically to the data encodings. First, we start by developping a REST interface for each service (SOS, SAS, SPS, and WNS). Each service can be implemented as a server application that encapsulates the SWE service instances and works as a proxy to this service for providing a RESTful interface to the data. For instance, Figure 3 shows the design principle of the most important service which is the SOS service. The RESTful SOS acts as a proxy to the actual SOS and transforms the input calls to SOS queries of the *GetObservation* operation. The proxy is also able to transform the observations encoded in the Observations ans Measurements (O&M) format to JSON data encodings, which is a lightweight and platform independent data exchange format. Parsing and creating the JSON representation of data requires less overhead on resource constrained sensor nodes compared to the XML format.

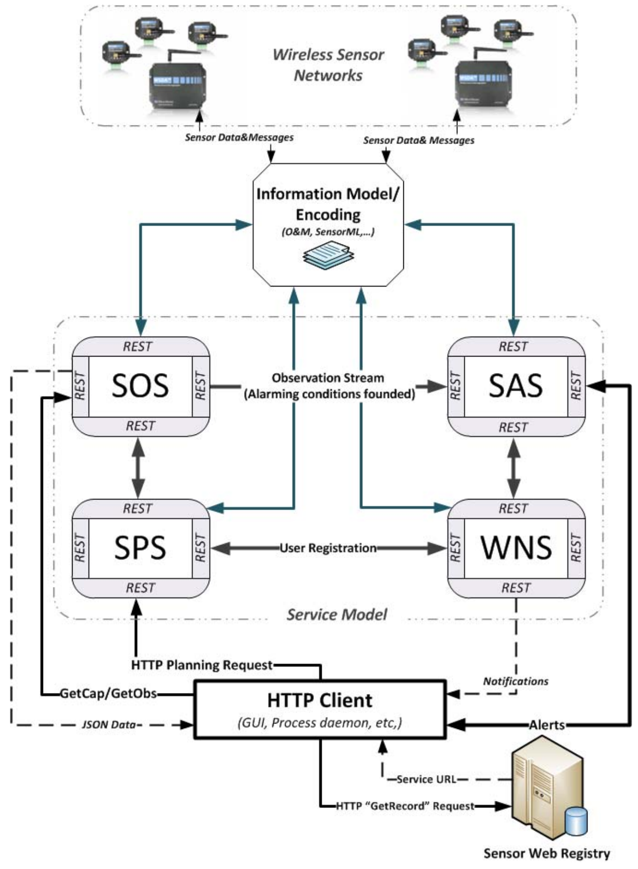
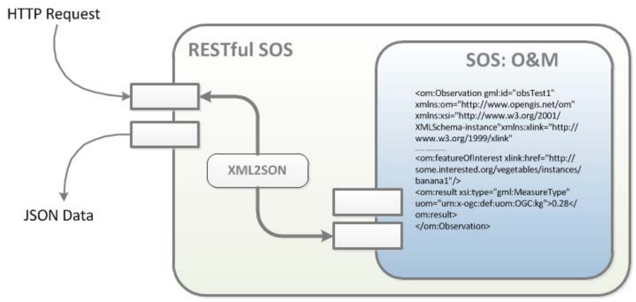


Figure 2. REST architectural style for SWE

In Figure 2, each HTTP client can be an application that requests for a sensor data/metadata or event or service. The request will be transmitted via a framework to the specific sensor node, which issues the response through its different OGC services. Each sensor node has a unique ID,by which it can join the network, advertise its services, provide its data and send and receive data using SWE standards (SOS, SPS, SAS and WNS) and REST operations (Get, Post, Put, Delete, etc).

The Web Registry Service (WRS) works as a middleware for connecting the client to the requested service. It is considered as a database server that receives the *GetRecords* request and returns a JSON document containing the endpoint URL of all existing services that satisfy the query. In a typical scenario, after getting the URL of the service, the client transmits a *GetFeasbility* request (includes the desired parameters if it is a Post request) to the Sensor Planning Service (SPS). If the task is feasible, the Get/Post observation request (with its parameters and the UserID) will be submitted through the SPS to the mismatched sensors to be executed. By submitting the request, the SOS starts collecting the sensed observations and stored them. Upon the task is completed, the SPS notifies/alerts the client that the requested data (events) is available (is executed). Then, the client can get observations/events from the SOS that was used to collect the desired data.



```
<om:Observation gml:id="obsTest1"
xmlns:om="http://www.opengis.net/om"
xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"xmlns:xlink="http://
www.w3.org/1999/xlink"
............
<om:featureOfInterest xlink:href="http://
some.interested.org/vegetables/instances/
banana1"/>
<om:result xsi:type="gml:MeasureType"
uom="urn:x-ogc:def:uom:OGC:kg">0.28</
om:result>
</om:Observation>
```

Figure 3. RESTful SOS

The registration of a new service in the WRS is done as follows. The client sends a request to the WRS containing the service URL to be registered. After registration, the WRS connects to the specified service, fetches its capabilites and starts processing it. When this operation is completed, the WRS notifies the client by sending a notification message to announce that the added service becomes searchable through the WRS interface.

The conversion of XML encodings to JSON format is done automatically using the developed convertor. To made a lossless and efficient converter, we have specified some rules to be used in the validation process. Moreover, in order to reduce the size of the JSON output string buffer, we have made an optimization that consists in ignoring all extraneous formatting spaces and tabulations during the creation of the JSON structure. Figures 4 and 5 present a fragment of an example of OGC observations and measurements and its corresponding in JSON format, respectively.

Then, since the manual generation of JSON objects would be a huge overhead and prone to error, an additional library has been developed to support the reformatting of a response as a JSON object. The JSON library helps to create JSON objects by offering services which automatically generate the JSON envelope. As will be proved in next section, this has an important impact of the parsing time and the size of the output file.

```
<om:Observation gml:id="obsTest1"
xmlns:om="http://www.opengis.net/om"
xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:gco="http://www.isotc211.org/2005/gco"
xmlns:gmd="http://www.isotc211.org/2005/gmd"
xmlns:gml="http://www.opengis.net/gml"
xsi:schemaLocation="http://www.opengis.net/om
../om.xsd">
<gml:description>Observation test instance<
/gml:description>
<gml:name>Observation test 1</gml:name>
<om:time>
<gml:TimeInstant gml:id="ot1t">
<gml:timePosition>2005-01-11T16:22:25.00
</gml:timePosition>
</gml:TimeInstant>
</om:time>
<om:location>
<gmd:EX_GeographicDescription>
<gmd:geographicIdentifier>
<gmd:MD_Identifier>
<gmd:code>
<gco:CharacterString>Subiaco Markets
</gco:CharacterString>
</gmd:code>
</gmd:MD_Identifier>
</gmd:geographicIdentifier>
</gmd:EX_GeographicDescription>
</om:location>
<om:procedure xlink:href=
"urn:x-ogc:object:feature:Sensor:OGC:scales"/>
<om:observedProperty xlink:href=
"urn:x-ogc:def:phenomenon:OGC:mass"/>
<om:featureOfInterest xlink:href=
"http://some.interested.org/vegetables/instances/
banana1"/>
<om:result xsi:type="gml:MeasureType"
uom="urn:x-ogc:def:uom:OGC:kg">0.28</om:result>
</om:Observation>
```

Figure 4. XML fragment of OGC observations and measurements

IV. Implementation and Evaluation

As a middleware system for building Sensor Web infrastructures based on SWE, we have chosen the open source software initiative 52 °North Sensor Web framework [4]. This middleware provides implementations for the different SWE services through the OGC Web Service Access Framework (OX-Framework [5]). The aim of the OX-Framework is to provide an integrative view to access all kinds of OGC Web services and visualize and process the required data. Several important applications are built on top of the OX-Framework, such as Thin SOS Client, uDig Plugin, and Web Map Server [19]. In our work, we have used the Thin SOSClient to interact with the SOS service. This application provides access to sensor data via the user interface of a thin Web

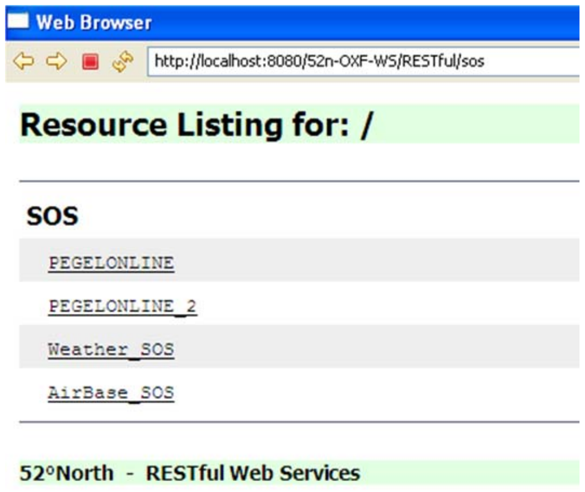
[4]http://52north.org/.

[5]http://52north.org/communities/sensorweb/oxf/index.html

```
{
"Observation":
{"id": "obsTest1",
"schemaLocation":
"http://www.opengis.net/om ../om.xsd",
"description": "Observation test instance",
"name": "Observation test 1", "time":
{"TimeInstant": {"id": "ot1t",
"timePosition": "2005-01-11T16:22:25.00"}},
 "location": {"EX_GeographicDescription":
{"geographicIdentifier":
{"MD_Identifier": {"code": {"CharacterString":
"Subiaco Markets"}}}}}, "procedure": {"href":
"urn:x-ogc:object:feature:Sensor:OGC:scales"},
"observedProperty":{"href": "urn:x-ogc:def:
phenomenon:OGC:mass"},"featureOfInterest":
{"href": "http://some.interested.org/
vegetables/instances/banana1"},"result":
{"type": "gml:MeasureType", "uom":
"urn:x-ogc:def:uom:OGC:kg", "#text": "0.28"
}}}
```

Figure 5. The corresponding JSON to the previous XML fragment of OGC observations and measurements

client runnable in a common Web browser. As a test scenario, we have considered the application presented in https://svn.52north.org/svn/swe/incubation/OXRestWS/trunk/OX-RestWS/ and extended it by considering the JSON exchange format. We have also used a sensor environment example described in http://v-swe.uni-muenster.de:8080/52nRESTfulSOS/RESTful/sos/AirBase_SOS/ that we have extended with new RESTful implementation of the other services. Figure 6 shows the RESTful SOS homepage that uses the SOS instances deployed at the Institute for Geoinformatics of the University of Muenster like PEGELONLINE, Weather_SOS and AirBase_SOS.



Resource Listing for: /

SOS

PEGELONLINE

PEGELONLINE_2

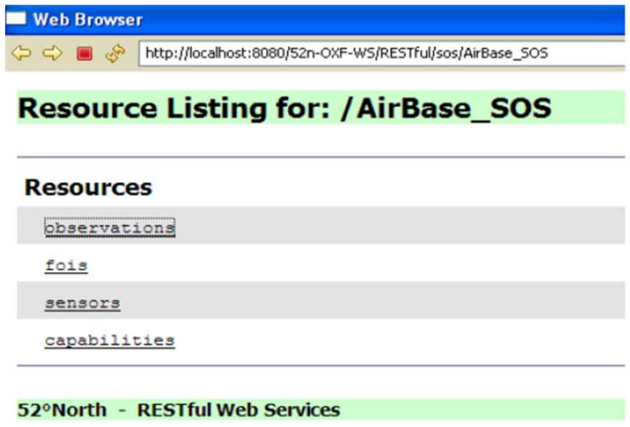
Weather_SOS

AirBase_SOS

52°North - RESTful Web Services

Figure 6. The OX-RESTful SOS hompage

As shown in figure 7, accessing to one of these SOS instances such as AirBase_SOS enables the client to discover the available resources (such as Observations and Sensors) and services (like Features Of Interest (FOIs) and Capabilities) of this instance. The response to the O&M request can be returned in a JSON format through our converter.



Resource Listing for: /AirBase_SOS

Resources

observations

fois

sensors

capabilities

52°North - RESTful Web Services

Figure 7. Available AirBase_SOS resources

We studied our test scenario benchmarking by using Apache JMeter [6]. For the benchmarking we have considered two parameters, which are the Data buffer size and the Transmission time. The results of measurements and performance studies show gains up to 60% and 15% in terms of transmission time and buffer size respectively.

To have an idea about the gain in term of data buffer size, figure 8 gives several examples was used in our tests.

XML	JSON	Gain
861	454	407
1240	978	262
1379	889	490
2133	1063	1070
2212	1146	1066
2394	1912	482
3417	1452	1965
12780	10300	2480
25474	20552	4922

Figure 8. Data Buffer Size Gain (Bytes)

The gain in terms of transmission time is shown in figures 9 and 10. Figure 9 gives a comparison between data transmission duration by considering:

- SOAP and XML
- REST and XML
- REST and JSON

We can observe a gain up to 60% in term of data transmission duration. It is clear that the reduction of the

[6]http://jakarta.apache.org/jmeter/.

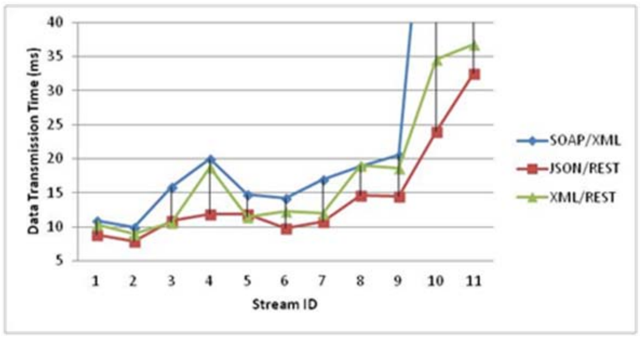


Figure 9. Data transmission duration - HTTP/REST VS SOAP/XML VS XML/REST

transmission time is more important by using REST instead of SOAP. Even for REST, with JSON format the results are better then with XML since XML is verbose.

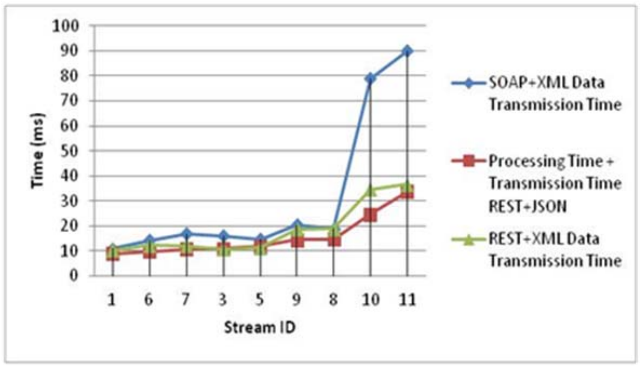


Figure 10. SOAP/XML Transmission time VS REST/JSON Processing+Transmission time

In figure 10, we consider the transmission time and the processing time together. In this figure, the gain is also important. The processing time is the time for parsing XML files to JSON ones. This time is negligible compared to the transmision time.

V. Conclusion and Future work

This paper depicted the adaptation of a RESTful architecture for OGC' s SWE services to overcome the SWE gaps. These gaps are mainly related to the data format and the architectural style followed by the implementation of the SWE services. This was motivated by the fact that sensor nodes can be viewed as RESTful resources that can be accessed and polled over the SWE services. Moreover, exchanging messages using the lightwight JSON data format instead of the XML one can trigger a good extension to the SWE intiative.

The performance evaluation results have showed the effectiveness of our RESTful architecture as well as the efficiency of adopting the JSON format in terms of file size reduction and communication time.

The approach presented here can be considered as a first step towards the work on SWE. Several open challenges and future work in this context can be outlined. Among these challenges, we are more interested in the improvement of interoperability, the facilitation of sensor and service integration, and the enablement of the Semantic Sensor Web. Indeed, the ability to dynamically integrate sensors is still an unresolved challenge within SWE. An on-the-fly integration of sensors into the Sensor Web with a minimum of human intervention is not straight-forward with the given methods, especially in hazard or disaster situations.

REFERENCES

[1] Arne Brring, Johannes Echterhoff, Simon Jirka, Ingo Simonis, Thomas Everding, Christoph Stasch, Steve Liang, and Rob Lemmens. New generation sensor web enablement. *Sensors*, 11(3):2652–2699, 2011.

[2] Cesare Pautasso and Erik Wilde. Restful web services: principles, patterns, emerging technologies. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 1359–1360, New York, NY, USA, 2010. ACM.

[3] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of json and xml data interchange formats: A case study. In *CAINE*, pages 157–162, 2009.

[4] Daniel Mandl, Pat Cappelaere, Stuart Frye, Rob Sohlberg, Lawrence Ong, Steve Chien, Daniel Tran, Don Sullivan, Stefan Falke, Steve Kolitz, and et al. Sensor web 2 . 0 : Connecting earth s sensors via the internet. *Earth Science*, 2008.

[5] Jeffrey Shneidman, Peter Pietzuch, Jonathan Ledlie, Mema Roussopoulos, Margo Seltzer, and Matt Welsh. Hourglass: An infrastructure for connecting sensor networks and applications. In *Harvard Technical Report TR-21-04*, 2004.

[6] Karl Aberer, Manfred Hauswirth, and Ali Salehi. A middleware for fast and flexible sensor network deployment. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 1199–1202. VLDB Endowment, 2006.

[7] José Cecílio and Pedro Furtado. Distributed configuration and processing for industrial sensor networks. In *Proceedings of the 6th International Workshop on Middleware Tools, Services and Run-time Support for Networked Embedded Systems*, MidSens '11, pages 4:1–4:6, New York, NY, USA, 2011. ACM.

[8] Luciana Moreira Sá de Souza, Patrik Spiess, Dominique Guinard, Moritz Köhler, Stamatis Karnouskos, and Domnic Savio. Socrades: A web service based shop floor integration infrastructure. In *Internet of Things 2008 Conference, Zurich, Switzerland*, volume 4952 of *Lecture Notes in Computer Science*, pages 50–67. Springer, 26–28 March 2008.

[9] Sanja Bogdanovic-Dinic Natasa Veljkovic and Leonid Stoimenov. Ginissense - applying ogc sensor web enablement. *Earth Science*, 2010.

[10] Xingchen Chu and Rajkumar Buyya. Service Oriented Sensor Web. pages 51–74. 2007.

[11] Tomasz Kobialka, Rajkumar Buyya, Christopher Leckie, and Ramamohanarao Kotagiri. A sensor web middleware with stateful services for heterogeneous sensor networks.

[12] Dzenana Muracevic, Fahrudin Orucevic, and Haris Kurtagic. Article:method of integration of geospatial data with business intelligent systems based on services oriented architecture. *International Journal of Computer Applications*, 7(7):35–39, October 2010. Published By Foundation of Computer Science.

[13] Dogan Yazar and Adam Dunkels. Efficient application integration in ip-based sensor networks. In *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, BuildSys '09, pages 43–48, New York, NY, USA, 2009. ACM.

[14] Lars Schor, Philipp Sommer, and Roger Wattenhofer. Towards a zero-configuration wireless sensor network architecture for smart buildings. In *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, BuildSys '09, pages 31–36, New York, NY, USA, 2009. ACM.

[15] Filippo De Stefani, Paolo Gamba, Emanuele Goldoni, Alberto Savioli, Davide Silvestri, and Flavio Toffalini. Renvdb, a restful database for pervasive environmental wireless sensor networks. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems Workshops*, ICDCSW '10, pages 206–212, Washington, DC, USA, 2010. IEEE Computer Society.

[16] Mohsen Rouached, Shafique Chaudhry, and Anis Koubaa. Lowpans meet service-oriented architecture. *JUSPN*, 1(1):39–48, 2010.

[17] Nader Mohamed and Jameela Al-Jaroodi. A survey on service-oriented middleware for wireless sensor networks. *Service Oriented Computing and Applications*, 5(2):71–85, 2011.

[18] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 805–814, New York, NY, USA, 2008. ACM.

[19] Arne Bröring, Eike H. Jürrens, Simon Jirka, and Christoph Stasch. Development of Sensor Web Applications with Open Source Software. In *OpenSoruce GIS 2009*. University of Nottingham, Suchith Anand, June 2009.