A WEB SERVICE APPROACH FOR LINKING SENSORS AND CELLULAR SPACES

U.Isikdag[a,*]

[a] CSD, University of Central Lancashire, UK
uisikdag@gmail.com

Commission II WGII/2

KEY WORDS: Web Services, Sensors, 3D, Spaces

ABSTRACT:

More and more devices are starting to be connected to the Internet. In the future the Internet will not only be a communication medium for people, it will in fact be a communication environment for devices. The connected devices which are also referred as Things will have an ability to interact with other devices over the Internet, i.)provide information in interoperable form and ii.)consume /utilize such information with the help of sensors embedded in them. This overall concept is known as Internet-of-Things (IoT). This requires new approaches to be investigated for system architectures to establish relations between spaces and sensors. The research presented in this paper elaborates on an architecture developed with this aim, i.e. linking spaces and sensors using a RESTful approach. The objective is making spaces aware of (sensor-embedded) devices, and making devices aware of spaces in a loosely coupled way (i.e. a state/usage/function change in the spaces would not have effect on sensors, similarly a location/state/usage/function change in sensors would not have any effect on spaces). The proposed architecture also enables the automatic assignment of sensors to spaces depending on space geometry and sensor location.

1. INTRODUCTION

In near future, increasing number of connected devices will make use of sensors and actuators to acquire information from their surrounding environment and to act accordingly. The connected devices (which are also referred as Things) will have an ability to interact with other devices over the Internet, provide information in interoperable form and consume/utilize such information. This overall concept is known as Internet-of-Things (IoT).

Web Services was the buzz-word of last ten years, as they enabled message-based method invocation over the Internet, which mainly served for the purpose of enabling the utilization of Web Based Software. As most research/studies was focused on user-web or user-software interaction the heavyweight WS-* technologies (using standards, such as SOAP, WSDL, UDDI) worked smoothly. In fact as mentioned in Guinard et al (2011a) these technologies are too heavy and complex for interacting with the connected devices (i.e. Things). In fact RESTful architectural style (which will be elaborated on in the following section) provide significant advantages in i.) facilitating the interaction with Things, ii.) consuming the information provided by these Things.

In recent years there is a huge trend for development of tiny embedded Web Servers (even on cards and microchips) which are capable of providing information regarding the its container device. The information is generally provided in structured XML document or JavaScript Object Notation (JSON) formats by these tiny Web Servers. These formats are easy to read and interpret for both humans and computers/devices. As mentioned in Guinard et al (2011a) this makes it possible to interact with devices via Web Browsers and thus explore the world of smart Things with its many relationships (via links to other related Things). Dynamically generated real-world data on smart objects can be displayed on such "representative" Web pages, and then processed with Web 2.0 tools. This is generally referred as web-enablement of devices, and offers various opportunities from novice to expert developers who can build applications on these devices. In addition, as Web 2.0 enables and supports Mashups (i.e. unified representation of small pieces of information that is acquired from different resources) many web-enabled devices can provide real-time information for these Mashups. Paraimpu (2012) is an example of Mashup environments where a device can broadcast information directly to the web.

Various urban management tasks ranging from emergency response, indoor navigation to traffic flow monitoring benefits from the use of real-time information that is provided with devices that are connected to the Internet. In fact as the nature of the urban management domain is concerned with the real-world phenomena the key aspect of the information is being geo-referenced. In addition, in urban management domain most events occur in spaces which can be represented with discrete geometries. The discrete (and predefined) geometries can be used to represent a space that an event occurs, a space that is observed, or a space that exists in a certain period of time. The spaces can represent an arbitrary volume outdoors, a bounding box (a container space of a building) or a volume indoors that is represented by boundaries (i.e. walls, floors etc.). In this situation, a device can provide information regarding the discrete space that the device is located in, or can provide information regarding its own state. In the first option (which is the most commonly observed situation) the device should provide information on its container space, on the other hand the space should be aware of the devices that is contained in them.

The research explained in this paper elaborates on an architecture developed with the aim of linking spaces and sensors (embedded in devices) in a loosely coupled manner using a RESTful approach. The objective was making spaces aware of devices, and making devices aware of spaces in a loosely coupled way (i.e. a state/usage/function change in spaces would not have effect on sensors, similarly a location/state/usage/function change in sensors would not have any effect on spaces). In essence, the developed architecture enables the automatic assignment of sensors to spaces depending on space geometry and sensor location. When a change in the location of the device (that the sensor is mounted to) occurs, the device is automatically updated with the information regarding its container space, in addition if the device has moved from one space to another, the state of the previous and next spaces (and the list of sensors contained in them) gets updated automatically. Once this real time information about devices and spaces is available it can be used to facilitate emergency and disaster response activities, building facility management, etc.

The paper starts with a background on the meaning and notion of cellular spaces, and their relation to sensors. The background section then continues with providing information on the RESTful Web and RESTful Sensor Services. The following section starts with introducing the Sensor Service Framework proposed for linking the information coming from Things with the spaces that this information is provided from. An implementation in the Information Acquisition Layer of the Sensor Acquisition Framework, involves the utilization of RESTful web services in acquisition, processing and representation of information. The implementation covered the development of 2 REST interfaces which are elaborated in the the paper.

2. BACKGROUND

As mentioned above, linking sensors and spaces via Internet requires appropriate subdivision of spaces, sensor that can be located and interfaces which allow access to the measurements of the sensors.

2.1 Cellular Spaces and Sensors

Many researchers have investigated approaches for indoor space subdivision (Meijers et al 2005, Liu and Zlatanova 2012, Brown et al 2013). A solid theoretical framework for establishing connection between sensors and spaces is presented in OGC Discussion Paper Requirements and Space-Event Modeling for Indoor Navigation (Nagel et al, 2010). As discussed the localization of moving subjects and objects in 3D Spaces is one of the key issues of indoor navigation as there is no absolute positioning method such as GPS available so far for indoor navigation. Outdoor localization is possible through the use of GNSS (e.g. GPS), in fact it is also necessary to know the boundary conditions of its outdoor location for many application domains (ranging from traffic management to flight tracking).

Nagel et al, (2010) discusses the notion of cellular space to represent indoors. The proposition is representing the covered space of 3D solid objects as decomposed or subdivided into a finite set of partitions, i.e. into 3D elements that will be referred to as Cells. The Cellular Space itself is composed of cells, which represent the structurally smallest unit of the respective space. Cells can have different sizes or shapes and every cell of the respective space includes information relevant for the navigation model. As stated by the discussion paper some indoor navigation applications rely on a further spatial decomposition of rooms according to the mode of navigation, e.g., to represent navigable and non-navigable areas with respect to the capabilities and limitations of moving persons. Moreover, the partitioning of indoor space into smaller units may also be induced by limited propagation areas of sensor-based positioning systems, e.g., systems based on RFID tags, which do not cover the spatial extent of an entire room. Thus, a room can be geometrically fragmented into Cells, which again represent non-overlapping parts of the room. This fine-grained subdivision of space and its dual cell-to-cell representation enables more detailed escape planning routes in a fire escape scenario. Furthermore, these single cellular partitions can be individually addressed by sensor-based positioning and tracking systems to provide a more accurate location of moving subjects or objects. (Nagel et al, 2010). Similarly the outdoor space can also be represented by the same approach where 3D discrete spaces can be defined or a 3D solid object can be decomposed into partitions (i.e 3D cellular spaces). The spatial hierarchy between these spaces can be built based on containment relationships. These cellular spaces can be used to represent the coverage volume of the sensor/device (i.e. it can be the space of signal propagation or it can be a space that a sensor is acquiring information from). In the focus of this research, this paper will concentrate on the latter one.

2.2 The RESTful Web

Web services can be defined as components and resources that can either be invoked over the web or reached by standard web protocols using standard messages. Two styles of Web Services exist today: SOAP and REST. SOAP is a web service style based on using SOAP (Simple Object Access Protocol) protocol for exchanging XML formatted messages between the networks by using Hypertext Transfer Protocol (HTTP). On the other hand REST is an architectural style where the web service operates by calling various web-resources. The terms REST and RESTful web services have been coined after the PhD dissertation of Roy Fielding (Fielding, 2000). As explained by Pautasso (2008), REST was originally introduced as an architectural style for building large-scale distributed hypermedia systems. According to REST style, a web service can be built upon resources (i.e. anything that is available digitally over the web), their names (identified by uniform indicators, i.e. URIs) representations (i.e. metadata/data on the current state of the resource) and links between the representations.

As stated by Guindard et al (2011b) at the core of a RESTful architecture lies resources that are uniquely identified through Uniform Resource Identifiers (URIs). The Web is an implementation of RESTful principles as it uses URLs to identify resources and http as their service interface. Resources can have several representation formats (e.g. HTML, JSON) negotiated at run time using HTTP content negotiation. In a typical REST request, the client discovers the URL of a service it wants to call by browsing or crawling its HTML representation. The client then sends an HTTP call to this URL with a given verb (GET, POST, PUT, DELETE.), a number of options (e.g., accepted format), and a payload in the negotiated format (e.g., XML or JSON). In regular browsers GET is used to request a resource (e.g. web page, picture, video from a server), POST is used to send a piece of data to the server side (without notifying the client about the data sent). Regular browsers do not make use of PUT and DELETE methods of HTTP, which can be invoked by specific REST clients/APIs that are developed for communicating with all HTTP methods. Several recent research projects implement RESTful Web services for smart things within (what has become to be known as) the Web of Things.

As mentioned earlier, interconnected devices (i.e. Things) today are capable of providing real time information about them on the web. The information can be acquired from various sensors that are connected to the device and broadcasted through the embedded web servers. The information broadcasted by these servers is usually in form of XML or JSON documents. Several loosely coupled web architectures can be designed to reach and utilize this information, where,

- Each sensor is represented by an URI (Sensor Endpoint)
- Each device is represented by an URI (Device Endpoint)
- A facade that interacts with devices is represented by and URI (Facade Endpoint).

In such situation, web services can be used to interact with the (Sensor / Device / Facade) endpoints. Due to its lightweight and loosely coupled nature, REST architectural style is preferred for device to web and web to device interactions. In recent research REST is viewed as ideal candidate to build a "universal" architecture and Application Programming Interface (API) for smart things. The use of Facade (i.e. wrapper layer) is recommended for communicating and interacting with devices and sensors. Usländer et al., (2010) suggest that Virtual Sensors (i.e. soft-sensors that are used to gather and abstract data from diverse sets of sensor network nodes) can act as a middleware between the Sensors and Services.

The client side of the architecture as indicated in Usländer et al. (2010) can be composed of visualisation, reporting and other sensor applications. In terms of visualisation of geospatial information, recent tools offer a huge potential. Since embedded web servers in an Internet of Things architecture generally have fewer resources than Web clients, browsers or mobile phones, Asynchronous JavaScript and XML (AJAX) clients has proven to be a good way of transferring some of the server workload to the client (Guinard et al, 2011a). As stated by Abernathy (2011), the advent of Web 2.0 tools such as AJAX have given online mapping tools increased versatility and visual appeal. Virtual globes such as Google Earth provide users access to geospatial information in three dimensions, allowing them to turn on and off various layers, download datasets, and create their own spatial data complete with text, photos, Web links, and even video clips.

3. THE WEB SERVICE

Isikdag and Zlatanova (2011) proposed a Sensor Acquisition Framework, as a web service architecture for information acquisition from indoor sensor networks, and for uniting information acquired from Things with 3D building model representations in the geospatial environment. According to the framework once the information from the indoor sensors has been acquired, this information can be visualised together with the 3D representations of Building Information Models, building representations in digital city models.

The Sensor Acquisition Framework concentrates on two dimensions, the first one being defining a web service architecture for provision of sensor information, the second one being acquisition and visualisation of this information in the geospatial environment. There also is a third dimension as the representation of acquired sensor information within building and indoor models, which also covers the design of models for representation of sensor's coverage (i.e. as sensor spaces) within the buildings.

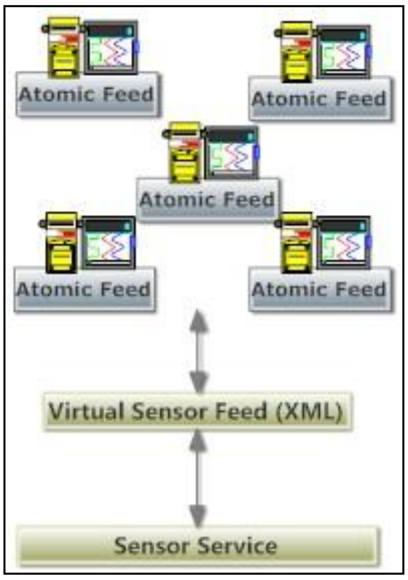


Figure 1. Information Acquisition Layer

As presented in Figure 1, the Information Acquisition Layer of the framework consists of connected devices (containing sensors and embedded web servers). These embedded web servers are able to publish lightweight XML feeds where each contain information about a single physical value (i.e. temperature) or multiple physical values (i.e. humidity and oxygen level). A virtual sensor is a software component that is able to collect information from other sensors, abstract and analyse this information and present it as if it is presented by a real sensor or mote. Information acquired from multiple sensors can be represented as the feed of the virtual sensor. For instance a virtual sensor can be used to present the average heat of a building by combining values acquired from different sensors.

The sensor service (i.e. WS-* or REST) will then be used to present the information.

The second layer of the framework deals with the representation of the information acquired from the sensor services and its visualisation. The framework focuses on three types of information consumers, the first set of consumers are Client Applications including Virtual Globes, Energy Monitoring Applications, software used in emergency response including GIS, Building Automation software, FM Software and so on. The second set of consumers would be the Web Portals. Currently portals such as Xively (Xively, 2013), ThingSpeak (ThingSpeak, 2011), Paraimpu (Paraimpu,2012) are becoming popular in representing the information coming from various sensors around the world. The third set of consumers of the information provided by, sensor services and digital building models would be personalized smart meter monitoring systems i.e. as explained in Kamilaris et al, 2010 and OpenMUC (OpenMUC, 2012) is another key software framework in the field.

3.1 Design and Implementation

The web service that will be elaborated in this section is developed as an implementation of the Information Acquisition Layer of the Sensor Acquisition Framework. The aim of the development was acquiring information from the feeds of virtual sensors, matching every virtual sensor with its container space based on the geo-location of the virtual sensor and space and then presenting this information through a consumer (e.g. a Virtual Globe).

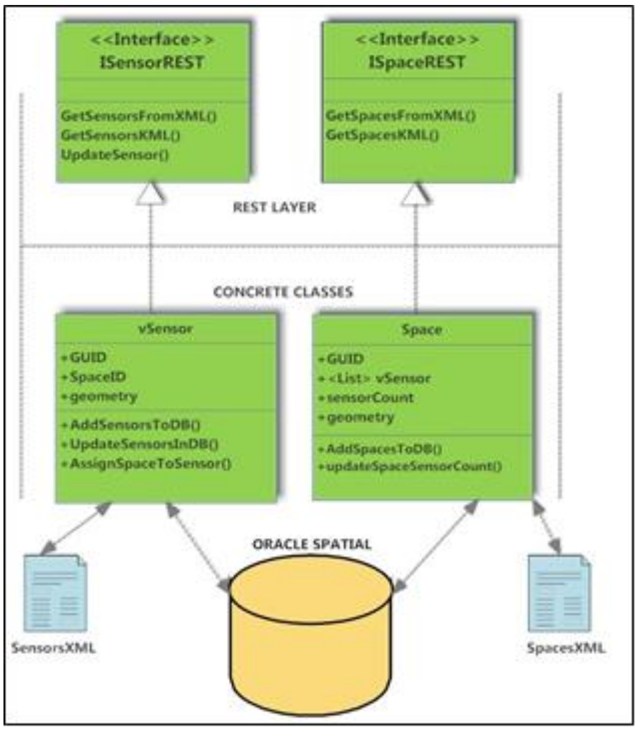
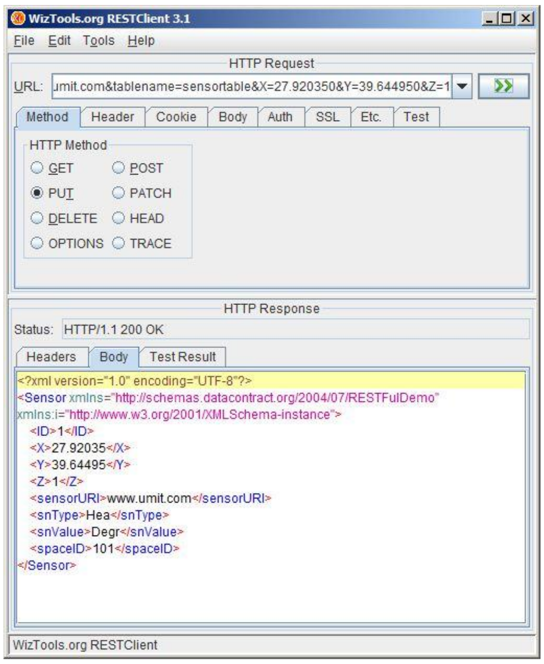


Figure 2.Web Service Framework

The web service architecture that is developed (Figure 2) utilizes the principles of REST. There are two RESTful interfaces that enable interaction through HTTP GET and HTTP PUT methods, there are two concrete classes (vSensor-virtual sensor and Space) that are used to implement the application logic. The application logic runs on the data layer that consists of an Oracle Spatial Database and XML files. The following summarizes the key methods and attributes of the implementation, based on interface components.

ISensorREST: This interface contains the definitions of methods for getting the information about virtual sensors from an XML file (Sensors XML) and generating required vSensor objects in the Oracle Spatial database, generation of KML representation of vSensor classes and updating the container of sensors in an event of location change. The concrete class vSensor deals with the application logic based on the methods of the *ISensorREST* , contains GUID , spaceID and geometry (related X,Y,Z) as attributes. The geometirc representation of vSensor is a 3D Point. The spaceID attribute contains the GUID of the Space that the vSensor is currently located in. The methods in the *ISensorREST* interface are implemented in vSensor concrete class. Once implemented in the concrete class vSensor, GetSensorsFromXML ( ) method interprets the information acquired from the Sensors XML file, generate a number of vSensor objects, and stores these objects in the Oracle Spatial database. AddSensorsToDB ( ) method is used to insert these objects into the database. The UpdateSensor ( ) method is used for updating the location or values of the sensors in an event of location or value change. This method calls AssignSpaceToSensor method to determine the new container (Space) of the vSensor after the location change. A 3D query in Oracle Spatial is executed to determine the new container space of the sensor. Once this is determined, the UpdateSensorsInDB ( ) method is called to update the sensor instance in the database with the new location, value and container. Once invoked through HTTP PUT request UpdateSensor ( ) method returns an XML representation of the vSensor that is updated (Figure 3.). The SpaceID attribute of the vSensor class is also updated in this stage.



URL: umit.com&tablename=sensortable&X=27.920350&Y=39.644950&Z=1

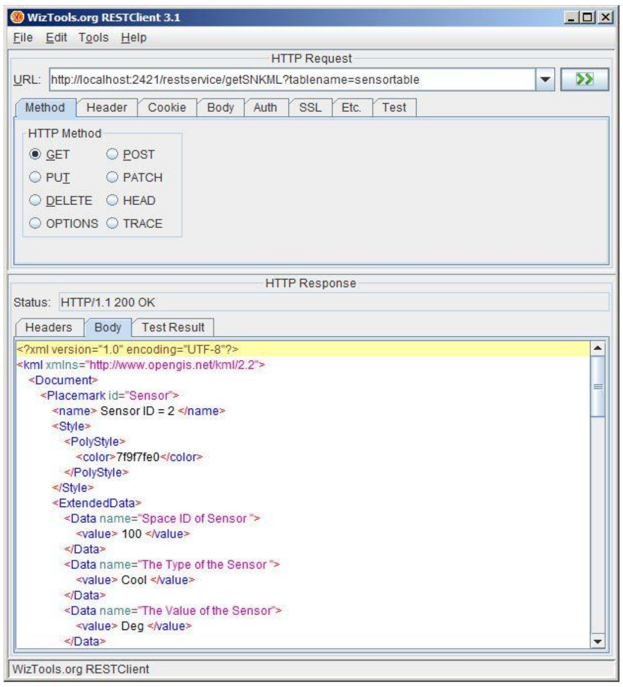
HTTP Method: ( ) GET ( ) POST [✓ PUT] ( ) PATCH ( ) DELETE ( ) HEAD ( ) OPTIONS ( ) TRACE

Status: HTTP/1.1 200 OK

```
<?xml version="1.0" encoding="UTF-8"?>
<Sensor xmlns="http://schemas.datacontract.org/2004/07/RESTFulDemo"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
   <ID>1</ID>
   <X>27.92035</X>
   <Y>39.64495</Y>
   <Z>1</Z>
   <sensorURI>www.umit.com</sensorURI>
   <snType>Hea</snType>
   <snValue>Degr</snValue>
   <spaceID>101</spaceID>
</Sensor>
```

Figure 3. XML Representation of the updated vSensor as HTTP RESPONSE

The method GetSensorsKML ( ) once implemented returns a KML representation of the vSensor class. The representation is generated as an extended version of data that is returned based on the Oracle Spatial's KML generation query. The utilities package of Oracle Spatial contains the function SDO_UTIL.TO_KMLGEOMETRY() to convert the instances in the database to a KML representation. The output generated as a result of a query in Oracle Spatial is then improved to represent the semantic information in form of KML <Extended Data>. The KML representation that is returned as a result of method invocation through HTTP GET request is illustrated in Figure 4.



URL: http://localhost:2421/restservice/getSNKML?tablename=sensortable

HTTP Method: [✓ GET] POST PUT PATCH DELETE HEAD OPTIONS TRACE

Status: HTTP/1.1 200 OK

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <Placemark id="Sensor">
      <name> Sensor ID = 2 </name>
      <Style>
        <PolyStyle>
          <color>7f9f7fe0</color>
        </PolyStyle>
      </Style>
      <ExtendedData>
        <Data name="Space ID of Sensor ">
          <value> 100 </value>
        </Data>
        <Data name="The Type of the Sensor ">
          <value> Cool </value>
        </Data>
        <Data name="The Value of the Sensor">
          <value> Deg </value>
        </Data>
```

Figure 4. KML representation of vSensors as HTTP RESPONSE

The URI of the HTTP GET request can be defined as a Network Link in Google Earth and it can be updated on specific time intervals to show most up-to-date information regarding a vSensor. Such a representation tested during this study is shown in Figure 5.

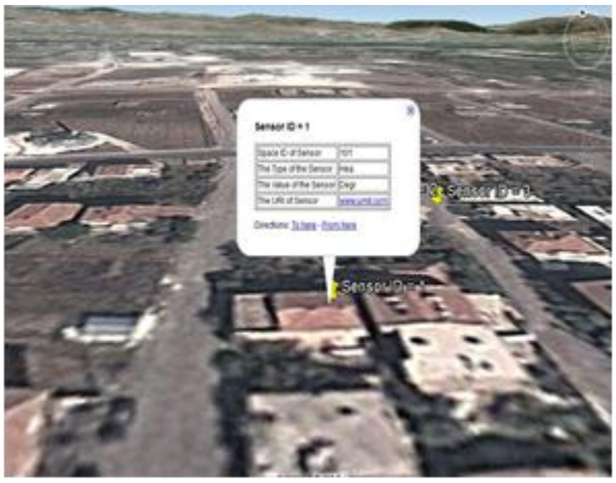


Figure 5. Visualisation of KML Representation of vSensor in Google Earth

ISpaceREST: This interface contains the definitions of methods for getting the information about Spaces from an XML file (Spaces XML) and generating required Space objects in the Oracle Spatial database, and generation of KML representation of Space classes. The concrete class Space deals with the application logic based on the methods of the *ISpaceREST* , contains GUID , vSensor List, SensorCount and geometry (related MinX, MinY, MizZ, MaxX, MaxY and MaxZ) as attributes. Every Space class maintain a list of vSensor(s) that it currently contains .The SensorCount attribute holds the number of sensors that is currently contained in that specific Space. The geometric representation of Space is a 3D Cell with Box representation. The corresponding Oracle Spatial representation is simple solid –defined by minimum and maximum coordinates. Similar to the vSensor, the methods in the *ISpaceREST* interface is implemented in Space concrete class. Once implemented in the concrete class, GetSSpacesFromXML( ) method invoked through HTTP GET request interprets the information acquired from the SpacesXML file, generate a number of Space objects, and stores these objects in the Oracle Spatial database. AddSpacesToDB( ) method contains the main application logic to store the Space objects in the database. The population of the <List> vSensor is achieved in a later stage with the invocation of GetSensorsFromXML ( ) method of the *ISensorREST* interface. Similarly the SensorCount attribute is updated through the GetSensorFromXML ( ) and UpdateSensor ( ) method calls in *ISensorREST* interface.

The method GetSpacesKML( ) returns a KML representation of the Space class. The representation is generated as an extended version of data that is returned based on the Oracle Spatial's KML generation query. The output generated as a result of Oracle Spatial query is then improved to represent the semantic information (i.e. the number of sensors per space) in form of KML <Extended Data>.The geometry of spaces is represented as a BRep model in KML (Figure 6).

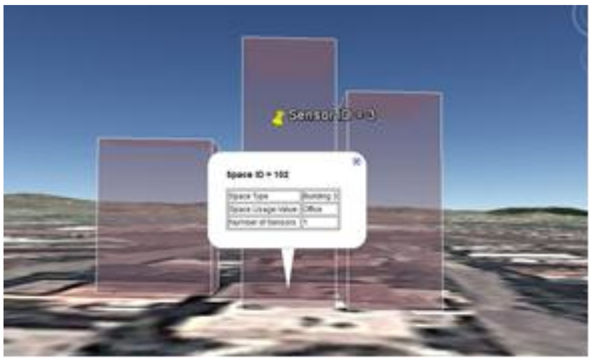


Figure 6. Visualisation of KML Representation of Space in Google Earth

4. SUMMARY AND CONCLUSION

The research presents and architecture for linking discrete spaces and sensors using a RESTful approach. The developed framework serves for the purpose of making spaces aware of (sensor-embedded) devices, and making devices aware of spaces in a loosely coupled way. The developed framework and its implementation demonstrated that RESTful web services can be used to; i.) obtain information from sensor feeds (i.e. in form of XML), ii.) persist this information in a spatial database, iii.) establish the containment relationship between sensors and spaces based on their geo-location and geometry, iv.) provide real-time information about sensors regarding their values, the space they are contained at a certain point in time, and v.) provide real time information on spaces including details on the sensors that are located in them at a certain point in time.Further research will focus on enhancing the developed interfaces and implementing them in real-life scenarios.

REFERENCES

Abernathy, D. 2011. Teaching the Geoweb: Interdisciplinary Undergraduate Research in Wireless Sensor Networks, Web Mapping, and Geospatial Data Management. *Journal of Geography*, 110(1), pp.27-31

Brown G., Nagel C., Zlatanova S.,and Kolbe T. H. 2013. Modeling 3D topographic space against indoor navigation requirements. In Pouliot, Daniel, Hubert & Zamyadi (eds.) Progress and New Trends in 3D Geoinformation Sciences Lecture Notes in Geoinformation and Cartography 2013, pp 1-22

Fielding, R. T. (2000). "Architectural styles and the design of network-based software architectures." PhD Thesis. Dept. of Information and Computer Science, University of California, Irvine.

Guinard et al (2011a) Dominique Guinard, Vlad Trifa, Friedemann Mattern, Erik Wilde From the Internet of Things to the Web of Things: Resource Oriented Architecture and Best Practices. In: Dieter Uckelmann, Mark Harrison, Florian Michahelles (Eds.): Architecting the Internet of Things. Springer, ISBN 978-3-642-19156-5, pp. 97-129, New York Dordrecht Heidelberg London, 2011

Guinard et al (2011b) Dominique Guinard, Iulia Ion, Simon Mayer, In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers' Perspective,Proceedings of Mobiquitous 2011 (8th International ICST Conference on Mobile and Ubiquitous Systems). pp. 326-337, Copenhagen, Denmark, December 2011

Isikdag, U. and S. Zlatanova, 2011, Sensor services for buildings: a framework and opportunities, In: Altan, Backhause, Boccardo&Zlatanova (Eds.), International Archives ISPRS XXXVIII, 7th Gi4DM, 3-7 May, Antalya, Turkey, 9 p.

Kamilaris, A., Trifa, V. and Guinard, D. (2010), "Building Web-based Infrastructures for Smart Meters", Energy Awareness and Conservation through Pervasive Applications Workshop http://www.cs.ucy.ac.cy/ResearchLabs/netrl/papers/files/KamilarisPervasive10.pdf

Liu, L. and S. Zlatanova, 2012, A sematic model for indoor navigation, ACM SIGSPATIAL ISA'12, Nov. 6, 2012. Redondo Beach, CA, USA, Copyright (c) 2012 ACM ISBN 978-1-4503-1697-2/12/11. Pp 1-8

Meijers, M., Zlatanova, S. and Preifer, N. 2005. 3D geoinformation indoors: structuring for evacuation, In Proceedings of Next generation 3D city models (Bonn, Germany, June 21-22, 2005

Nagel, C., Becker, T., Kaden, R., Li, K-J., Lee, J., Kolbe, T-H. 2010. Requirements and Space-Event Modeling for Indoor Navigation *OGC OpenGIS Discussion Paper* http://portal.opengeospatial.org/files/?artifact_id=41727(accessed 22.Jan.2011)

OpenMUC (2012) Available Online at http://www.openmuc.org

Paraimpu (2012) Available Online at http://paraimpu.crs4.it/

Pautasso,C., Zimmermann,O. and Leymann,F. (2008) "Restful web services vs. "big"' web services: making the right architectural decision" *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pp.805-814

Thingspeak(2011) Available Online at http://www.thingspeak.com

Usländer,T., Jacques,P., Simonis,I., Watson,K. 2011. Designing environmental software applications based upon an open sensor service architecture, *Environmental Modelling & Software*, 25(9), pp. 977-987

Xively (2013) Available Online at https://xively.com