RESOURCE ORIENTED ARCHITECTURE FOR MUTIMEDIA SENSOR NETWORKS

IWAIT2009

Hiroshi IWATANI, Masayuki NAKATSUKA, Yutaro TAKAYANAGI, Jiro KATTO

Dept. of Computer Science, Graduate School of Fundamental Science and Engineering

Waseda University

3-4-1 Okubo, Shinjuku-ku, Tokyo, JAPAN

E-mail: {iwatani, nakatsuka, takayanagi, katto}@katto.comm.waseda.ac.jp

ABSTRACT

Sensor network has been a hot research topic for the past decade and has moved its phase into using multimedia sensors such as cameras and microphones [1]. Combining many types of sensor data will lead to more accurate and precise information of the environment. However, the use of sensor network data is still limited to closed circumstances. Thus, in this paper, we propose a web-service based framework to deploy multimedia sensor networks. In order to unify different types of sensor data and also to support heterogeneous client applications, we used ROA (Resource Oriented Architecture [2]).

Keywords: multimedia sensor networks, resource oriented architecture, sensor fusion

1. INTRODUCTION

Human life style has changed a lot since the invention of mobile computers. Though it brought a lot of efficiency and convenience to the world it has also brought unnecessary displeasure. Displeasure such as ringing phones in a middle of meetings and funerals. Also the trouble of deciding what query to use for a search engine. Imagine a world where all machines read the situation and behaves itself to match the environment. There would be no cell phones that start ringing in the middle of meetings and the answer to our question would be what we want.

To realize such a world the most important information that needs to be retrieved is the environmental information, the information about the situation where the machine is. This will lead to the use of sensor network technology. If we can retrieve sound and visual information of the environment we could estimate the occasion of the environment. Thus cell phones could decide to ring or to vibrate. If we could retrieve temperature and humidity information we could get better answers for restaurant searching services.

1.1 Background

Multimedia Sensor Networks has become a hot research topic in the sensor network field [1]. However the research is still narrow and limited to certain cases. It is obvious that by combining different types of sensors and making a sensor fusion, the outcome of the information could be more accurate and precise.

The problem is combining many types of sensors would be confusing. Due to the advancement in multimedia sensors it is now cheap and light in power to use them for sensor networks. This leads to many types of data such as pictures, movies, sounds, temperatures, and so on.

Thus we propose the deployment of sensor data as web services. To unify different types of sensor data and also to support heterogeneous client applications, we used ROA (Resource Oriented Architecture [2]).

1.2 Related Work

We had been working on sensor network research such as [4][5]. [4] uses signal strength from sensor nodes to estimate the crowd density of a certain area. Also [5] uses camera sensors to estimate the position of a human being. It is obvious that these two researches could make up for each other if they used both the signal strength information and camera data information.

Since we are applying a unique URI to each sensor data our research could be related to [7]. It is a technology to manage information bound to a certain object or a location. It uses a so called *ucode* that is a unique ID assigned to every single objects and places. So all the information would be linked to the ucode and users can retrieve information using the ucode. One use case is by applying ucodes to RFID tag embedded objects. Users with mobile terminals would read the ucode from the tag and can automatically get information about the object.

In the next section we talk about ROA and applying sensor data to ROA. In section 3 we will mention on the implementation. In section 4 we will talk about matters that were brought up from the implementation and lastly in section 5 we finish with the summary.

2. RESOURCE ORIENTED ARCHITECTURE

2.1 Background

ROA was set up by Richardson, based on REST (Representational State Transfer [3]). Before it was established, the words SOAP (Simple Object Access Protocol) and REST swirled around to claim that they were better than the other [6]. The problem was both SOAP and REST were not exactly an architecture thereby created confusion. SOAP is a technology (protocol) often used for Service Oriented Architecture (SOA). It is an xml based envelope to put information that needs to be sent between the server and client in SOA web services. REST in the other hand was a way to evaluate architectures. If certain architecture fulfills a certain form than the other, then the architecture is more RESTful than the other. ROA was formed so that it scores good points when evaluated by its' RESTfulness.

Here we will give a brief explanation of methods to deploy web services. One of the ways is Remote Procedure Call (RPC) often used to realize SOA and the second way is REST-RPC hybrid and the third one is ROA.

~RPC~
Using RPC to deploy new services means the service is method-base. For example, when we want to create a blog entry search service it would be something like get_blogentry(). Deleting a blog entry might be delete_blogentry(), creating a blog entry might be post_blogentry(). This seems simple but many blog service providers may create their original services, leading to many methods: getblog(var), get_blog(var), get_article(var) and so on… To use a certain service from a single provider wouldn't be much trouble but when creating blog searching application from all of the blog service providers it would be troublesome. We need to check how to use each method and test them all.

~REST-RPC Hybrid~
This style was created in the consequence of the SOAP-REST argumentation. Instead of deploying methods REST-RPC Hybrid deploys URLs. Following the former blog service example, it would be something like http://example.com/get_blog. Arguments are connected to the URL using "?" and "&". The request is sent to the URI using HTTP method GET and the response is usually in XML format. Since the service has a URI we can check responses using web browsers, leading to the convenience compared to RPC. However the big problem is the mismatch with HTTP methods. The HTTP protocol offers methods not only GET but also POST, PUT, DELETE and many more. However REST-RPC Hybrid would only use GET which means deleting a blog entry would be as follows: *GET* http://example.com/*delete*blog?blog_id=1, an obvious mismatch.

~ROA~
ROA services, instead of deploying methods it deploys resources (some kind of information). Like the REST-RPC Hybrid, ROA uses URIs to name the resources. The difference is, it suggests to keep verbs out. So blog service example would be http://example.com/blog_entry/1. To use the service the 4 main HTTP methods (GET, POST, PUT, and DELETE) are used. GET is for retrieving the resource, POST is for creating new resources, PUT is for updating existing resources, and DELETE is for deleting the resources. (Also there are two other methods OPTION and HEAD but these are utility methods so we will not talk about it here.) To search a blog entry, it would be something like: GET http://example.com/blog_entry?q1=a&q2=b.

The reason to keep the methods to four basic methods is to keep complexity out. Notice that in the RPC example, there were many methods to the same blog searching service (getblog(), get_blog(), get_entry()…), in ROA there would be a lot of resources but there is no need to think about how to use them.

2.2 ROA and Sensor Data

2.2.1 Naming

Here we talk about the naming of the resources when we apply ROA to sensor data. Naming means how to structure the URIs. The two main rules are:

1. mainly composed by nouns and adjectives
2. structured and ordered from broad to precise: (http://example.com/A/B/C would mean A ⊃ B ⊃ C)

So for sensor data it would be something like http://example.com/sesordata/some_way_to_identify_each_node/sensor_type. One simple way to identify each would be a simple number. This however is not a practical way. One of the main aspects of ROA is that by looking at the URI, we can understand what the resource is. From the fact that sensor nodes exist in a physical form, it must be located somewhere. Thus we used location information to identify each sensor node (also strongly affected by [1]): http://example.com/sensordata/longitude,latitude/sensor_type. So the name of the resource for a camera sensor placed at (longitude: 139.77048055555556, latitude: 35.67777222222222) would be http://example.com/sensordata/139.77048055555556,35.67777222222222/camera.

2.2.2 Other Sensor Nodes

Though most sensors are placed at a certain spot, there are exceptions. One example would be sensors that are placed on movable objects. The naming should be something like http://example.com/sensordatas/dynamic/object_id/sensor_type. Same reason as mentioned before, object_id should be something that can identify the object so that a person can determine what the resource is from the URI. For example if it is a car, then it would be practical to use the car number.

Another exception is GPS sensors. We do not want to know the data of the GPS sensor if we know where it is. Thus like the dynamic sensors it should be something like http://example.com/GPS/object_id.

2.2.3 Retrieving the Nearest Resource

To improve usability the system should retrieve the nearest information from any spot. Thinking about an application on GPS sensor embedded mobile terminal, users would want to retrieve sensor data near them. Thus when a user asks for http://example.com/sensordatas/any_value_of_longitude/any_value_of_latitude/sensor_type it should give corresponding responses in order from nearest to far.

2.2.4 Methods

If we apply ROA to sensor data management web service system, then the four basic methods would mean:

・GET: Getting the sensor data. It would be mainly used from client applications.

・POST: Creating a new sensor data resource. It would be used when deploying a new sensor node.

・PUT: Updating sensor data resource. It would be mainly used from sensor nodes to keep its' data information updated. Also it would be used when sensor nodes were replaced to update its' location information.

・DELETE: Deleting the sensor data resource. It would be used when sensor nodes gets removed.

3. IMPLEMENTATION

For the implementation we used Ruby on Rails[8]. This framework is famous for its functionality to create network applications. The main reason to use this framework was because since its last major upgrade at the end of 2007, it supported RESTful architecture development.

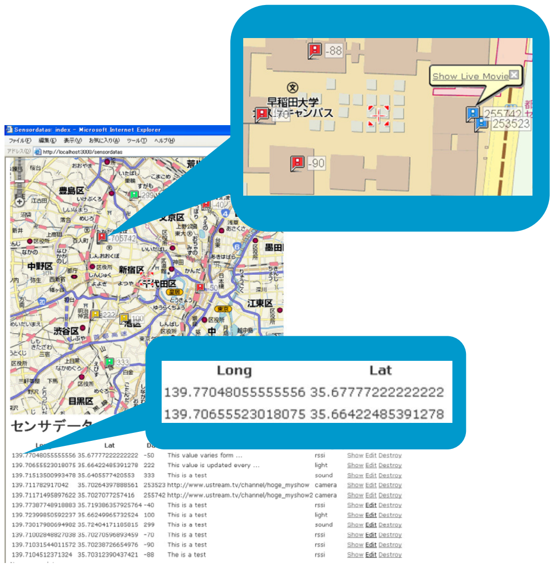
To retrieve longitude and latitude information and due to the deep relation between location and the resources, we mashed up the application with Yahoo!Maps[9]. Also to deploy camera sensor data, we used Ustream[10] which is a service that enables real time video broadcast over the web. The outcome of the implementations are shown on Fig.1, 2, and 3.

Besides the camera sensor we also implemented light and sound resources using MOTE[11] micaZ. First we created the resource from the web browser. Then we wrote a simple HTTP communication program in JAVA that sends an HTTP PUT method request to the resource with the newest value from the sensor as arguments.

4. OBSERVATIONS

From the implementation we can observe the merits listed below:
1. We can understand what the resource is by observing the URIs.
2. As long as HTTP communication is possible, the data could be accessed by any OS or any programming languages.
3. Checking the value could be done from the web browsers.

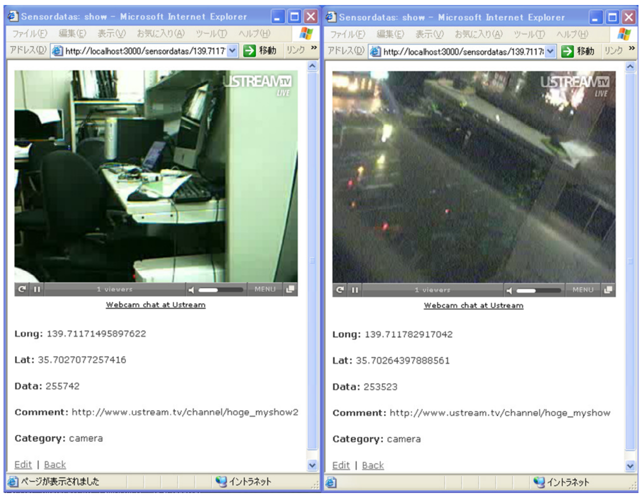
However there are matters to be thought of:
1. Security
2. Naming Matter



Long	Lat
139.77048055555556	35.67777222222222
139.70655523018075	35.66422485391278

	Lat				
139.77048055555556	35.67777222222222	-50	This value varies form ...	rssi	Show Edit Destroy
139.70655523018075	35.66422485391278	222	This value is updated every ...	light	Show Edit Destroy
139.71513500993478	35.6405577420553	333	This is a test	sound	Show Edit Destroy
139.711782917042	35.70264397888561	253523	http://www.ustream.tv/channel/hoge_myshow	camera	Show Edit Destroy
139.71171495897622	35.7027077257416	255742	http://www.ustream.tv/channel/hoge_myshow2	camera	Show Edit Destroy
139.77387748918883	35.719386357925764	-40	This is a test	rssi	Show Edit Destroy
139.72399850592237	35.66249965732524	100	This is a test	light	Show Edit Destroy
139.73017980694982	35.72404171185815	299	This is a test	sound	Show Edit Destroy
139.71002848827038	35.70270596893459	-70	This is a test	rssi	Show Edit Destroy
139.71031544011572	35.70238726654976	-90	This is a test	rssi	Show Edit Destroy
139.7104512371324	35.70312390437421	-88	The is a test	rssi	Show Edit Destroy

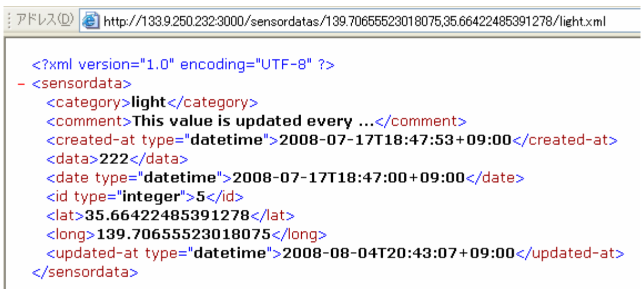
Fig. 1: Implementation.


Sensordatas: show - Microsoft Internet Explorer
http://localhost:3000/sensordatas/139.7117
Webcam chat at Ustream
Edit | Back
ページが表示されました
イントラネット
Sensordatas: show - Microsoft Internet Explorer
http://localhost:3000/sensordatas/139.7117
Webcam chat at Ustream
Edit | Back
イントラネット

Long: 139.71171495897622
Lat: 35.7027077257416
Data: 255742
Comment: http://www.ustream.tv/channel/hoge_myshow2
Category: camera

Long: 139.711782917042
Lat: 35.70264397888561
Data: 253523
Comment: http://www.ustream.tv/channel/hoge_myshow
Category: camera

Fig. 2: Example of camera resources.



http://133.9.250.232:3000/sensordatas/139.70655523018075,35.66422485391278/light.xml

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <sensordata>
    <category>light</category>
    <comment>This value is updated every ...</comment>
    <created-at type="datetime">2008-07-17T18:47:53+09:00</created-at>
    <data>222</data>
    <date type="datetime">2008-07-17T18:47:00+09:00</date>
    <id type="integer">5</id>
    <lat>35.66422485391278</lat>
    <long>139.70655523018075</long>
    <updated-at type="datetime">2008-08-04T20:43:07+09:00</updated-at>
  </sensordata>
```

Fig. 3: Response example for light sensor resource request.

4.1 Security

Management of the data would be important. Simple solution would be to manage them in user base form. By implementing authentication functions we can control the resources. Authority to use the PUT, POST and DELETE methods are given to only the authenticated users. By keeping the GET method open, everyone can still access the resources. Privacy is also a concern, especially data from the cameras. This problem is a big topic so we will not mention it here but mean while we have to keep in mind where to place the cameras so that unaware people's privacy wouldn't be violated.

4.2 Naming Matter

Though ROA clearly says to name the resources using nouns and adjectives, and to format it as a hierarchy, there is room for debate. In the implementation, we have followed the rules of ROA, but there could be better ways and still match the same rules. However looking at the system from the client side, I believe it is one of the best ways.

There are many ways to estimate location of users/mobile terminals. Simplest way is to use GPS sensors. More than 40% of the cell phones in Japan have GPS sensors embedded. Since the police department had trouble locating emergency calls from cell phones, GPS sensor embedded cell phones are going to grow its rate for sure. Another way would be to use base stations. An example would be PlaceEngine[12]. In a more local estimation there are ways to use camera sensors and other sensors too.

From the examples above, there are many ways to express a location. GPS sensors give the location using longitude and latitude. It may depend on services but base station method gives the location using addresses. Local estimation methods give its location in the form so that it identifies where it is within the local area. The only one way to cover them all I believe is longitude and latitude. There are services[13] that give longitude and latitude information from addresses. If we want the exact position for the local estimation location, then we can express it by simply extending the longitude, latitude values to more precise values. One of the merits web services have is its characteristics that it doesn't have constraints from the OS or programming languages. By managing data using longitude and latitude it would keep the information more common and have the most flexibility for many types of applications and terminals.

5. SUMMARY

In this paper we have proposed to deploy sensor data as Resource Oriented Architecture web services. We also implemented the system to check the serviceability. By naming the resources using location information the resources can be available for GPS sensor embedded mobile terminals. This will lead to realize more context aware machines and applications. Also there is big hope that new and innovative applications may be created since web services are open to the web and many people from different fields can access the data.

6. REFERENCES

[1] I. Akyildiz et al.: "Wireless Multimedia Sensor Networks: A Survey," IEEE Wireless Comm., Dec.2007.

[2] L. Richardson, S. Ruby. "RESTful Web Services." O'Reilly, 2007.

[3] Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, Dept. of Information and Computer Science, University of California at Irvine, 2000.

[4] M. Nakatsuka, J. Katto: A Study on Passive Crowd Density Estimation, ICMU2008 Tokyo, JAPAN

[5] Y. Takayanagi, J.Katto: "A Study on Real-Time Human 3D-Localization System" Technical Report of IEICE, Vol.108, No.127, pp.73-76, Jul.2008 (in Japanese).

[6] Michael Muehlen. Developing Web Services Choreograph Standards—the case of REST vs. SOAP. Decision Support Systems 40, 2005.

[7] Ubiquitous ID Technologies http://www.uidcenter.org

[8] Ruby on Rails http://www.rubyonrails.org/

[9] Yahoo!Maps http://developer.yahoo.co.jp/map/

[10] Ustream http://www.ustream.tv/

[11] Crossbow http://www.sensor-network.net/

[12] PlaceEngine http://www.placeengine.com/

[13] Geocoding http://www.geocoding.jp/