Sensor Overlays for Google Map

Xiuhan Hu (xh2234@columbia.edu)

December 31, 2014

1 Introduction

The Web of Tings (WoT) allow real-world objects to be part of the World Wide Web, and users can view or manipulate things easily via the Internet. To visualize sensors on a map is essential to achieve WoT, since we often need to know the remote environment in a panoramic view before we change specific things.

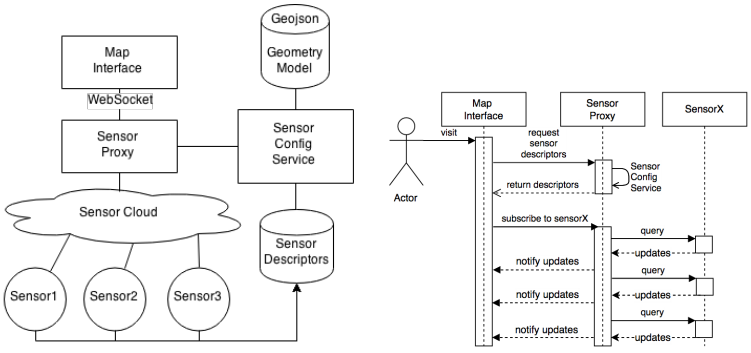
The goal of the project, Sensor Overlay for Google Maps, is to find a way to display information from the sensors as an overlay on Google Maps. The map view should be updated in real-time and work without user intervention most of time. There are innumerable kinds of sensors in real world, so the user should be able to customize the appearance according to the state of sensors. Another problem is that if there are too many sensors in a narrow space, the map view will be too crowded to be readable, unless we can cluster the sensors in a reasonable way. Existing solutions do not provide a realtime updating interface with automatically clustering function [2].

In section 2, the structure of overall system and the user interface facilities are described briefly, which explains the real-time updating feature of the map. Section 3 shows how a sensor descriptor is defined and how sensors can be rendered with different shapes, icons and colors. The solution for sensor clustering problem is covered in Section 4. Concluding remarks are given in Section 5 .

2 Overall Structure

The map interface need to provide real-time updated value of sensors. It should be able to communicate with sensors in certain way. However, it is costly for the interface to talk to individual sensors directly. So a sensor proxy service is created that the interface can establish a WebSocket connection to [Fig 1(a)]. The service provides basic information about a sensor as a sensor descriptor and it also provides an API for the interface to subscribe to the sensor. The following shows how the system will response to a viewer [Fig 1(b)]. 1) The map interface request all available sensor descriptors from sensor proxy service. 2) The sensor

proxy service will consult sensor config service, and the config service will collect information from sensors to build sensor descriptors, combine descriptors with their geometry models and return those descriptors to the map interface via sensor proxy service. 3) Then for each sensor, the map interface will subscribe to it through sensor proxy service. 4) The sensor proxy will keep querying the latest updates from subscribed sensors, and notify the interface of the updated values at intervals.



(a) Components (b) Sequence Diagram

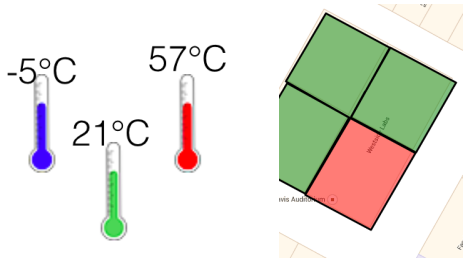
Figure 1: Overall Structure

Now the map interface can get the updated values of sensors.

3 Sensor Descriptor And Rendering

A sensor is rendered as a marker on the map interface, and its location or geometry is provided by its corresponding sensor descriptor. To generally describe all kinds of sensors, sensor descriptors should be well-defined in an abstract way.

We classify sensors into two general types [Fig 2]: environmental sensors and structural sensors.



(a) Environmental (b) Structural

Figure 2: Two types of sensors

An environment sensor provides information about the environment (ambient temperature, humidity, light intensity) in form of a number at certain location. A structural sensor reports on the state of a particular physical object (door open/close, lights on/off) in a certain shape recorded in our geometry model database.

We also want the appearance of the sensor marker to reflect the sensor's value. For example, the icon of thermometer marker will change from green to blue when it is getting too cold.

Taking account the above discussion, we borrow ideas from web design to build a sensor descriptor.

	Web	Sensor Descriptor
Look & Formatting	CSS	SensorStyle
Structure & Content	HTML	SensorGeometry & SensorData
Dynamic Altering	Javascript	SensorScript

Table 1: Sensor Descriptor components and their counterparts in web design

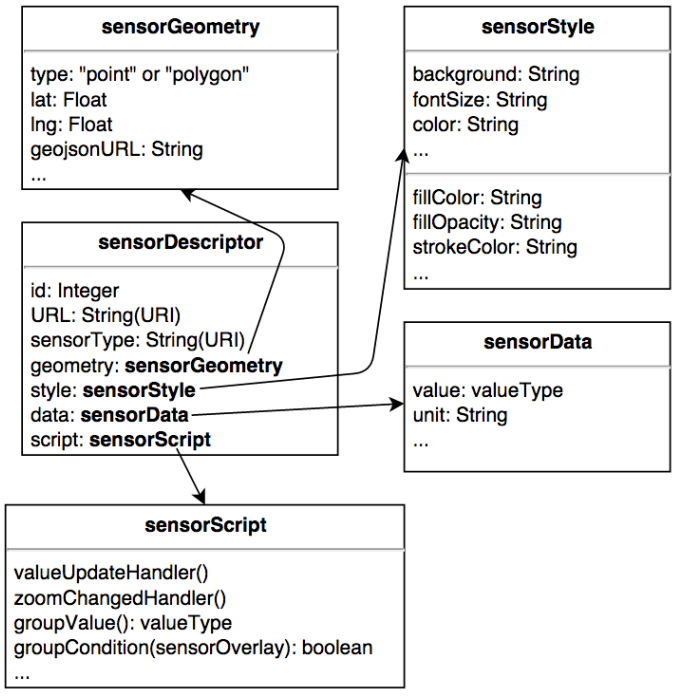


Figure 3: Sensor Descriptor Class Diagram

As shown in Figure 3, the *SensorStyle* part of *SensorDescriptor* describes

how the sensor marker looks like on the map interface. The *SensorGeometry* part describes the location and shape of the sensor, which can cover all kinds of geometry information need for both types of sensors. The *SensorData* gives the value and other relevant data for the sensor. The *SensorScript* specifies how a sensor marker will change according to a recent event. The loose coupled structure of a sensor descriptor enable us to reuse certain parts of it conviniently.

When the interface need to show a sensor on the map, it will initialize an instance of *SensorOverlay* from the given *SensorDescriptor*, which copies the information from the *SensorDescriptor* object to the new *SensorOverlay* object. Then that *SensorOverlay* object will draw itself on the map and register several event handler functions according to *SensorScript*.

To explain it more specifically, the following Figure 4 shows how valueUpdateHandler() will be invoked.

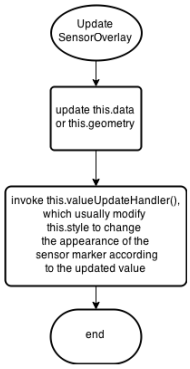


Figure 4: Sensor Update

The map interface will draw the sensor marker again after updating. The user will know there is a significant event (eg. temperature out of range) if a sensor marker changes its appearance obviously.

4 Sensor Clustering

If we have too many sensors to display in a small map area, the map will be unreadable for our users. Thus, automatically clustering sensors in that case is desirable for our map interface [1]. In *SensorScript*, we define groupCondition() function to determine whether a sensor should be clustered with another sensor. groupCondition() function accept another *SensorOverlay* object as input and return true if that sensor is getting too close to this sensor.

When the zoom level of the map is changed by user, we first randomly choose several delegate sensors. Then we check other sensors by groupCondition() function to cluster them with right delegate sensors, and hide them from the map view. After we get clusters of sensors, we can calculate the integrated value

of a cluster by a predefined function groupValue(). Then all *SensorOverlay*'s zoomChangedHandler() function fires to finish the clustering process.

Figure 5 shows how the map will response to a zoom level change event. For environmental sensors, we probably want to know the average value of a cluster of sensors, so the groupValue() for them should be an arithmetic average function. For structural sensors, for example the door open/closed sensors, we need to warn the user if any sensor is in abnormal state, so the groupValue() for them should be a logic OR operation. The groupValue() can be generally considered a reduce function for a set of sensor values.

Figure 6 shows how thermometers are clustered when zoom level is changed.

It is more complicated to visualize clustered structural sensors, we need to specify how we want to change the shape of delegate sensor map. In many cases, we can draw a convex hull covering all clustered sensors [Fig 7].

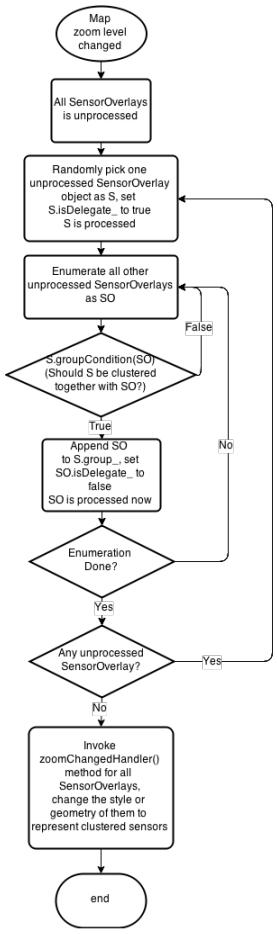


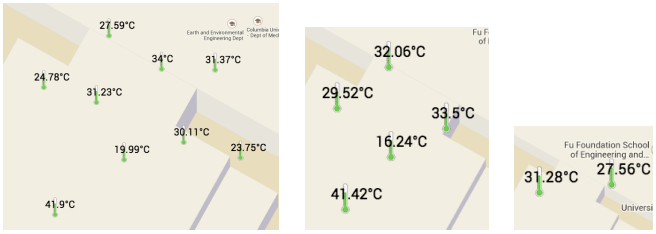
Figure 5: Zoom level changed

5 Conclusion

A sensor simulator instead of a real world sensor cloud is set for testing our map interface. The interface works well with current test setting and can provide clear and meaningful information for users. Reviewers can understand the interface with few explanation and no difficulties.

Through implementing the map interface, we have recognized that visualization should be semantic and information conveyed to user should be restrained. To improve our map, we can define more event handlers for sensor markers for a more versatile application.

In conclusion, the interface we designed is a solid step toward a general-purpose sensor visualization on a map view, which provide adequate information and a lighter burden of recognition for users.



(a) zoom=20 (b) zoom=19 (c) zoom=18

Figure 6: Enviromental sensor clustering

(a) zoom=20 (b) zoom=19 (c) zoom=18

Figure 7: Structural sensor clustering

References

[1] Cole Coleman and Peter Sklar. Clustering user interface, August 4 1998. US Patent 5,790,121.

[2] Suman Nath, Jie Liu, Jessica Miller, Feng Zhao, and Andre Santanche. Sensormap: a web site for sensors world-wide. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 373–374. ACM, 2006.