A Model and Framework for Reality-Oriented Web Services

Takahiro KOITA, Tatsuya Izumori* and Kenya SATO
Department of Information Systems Design,
Doshisha University, Japan

Abstract

Mobile devices that use information from sensors such as GPS and accelerometers are becoming widespread today, and web services using a variety of sensor information have appeared. This paper refers to these web services as reality-oriented web services. Since existing frameworks not designed to handle sensor information are useless when developing reality-oriented web services, high development costs are a problem. This paper proposes a model for configuring reality-oriented web services, making a comparison with existing frameworks. We discuss the design of a framework using this model and study its application to an actual web service.

Keywords: Google Maps, Ubiquitous Network, Ajax, Distributed Device Control

1. INTRODUCTION

1.1 Background
Web services that use the sensor functions of mobile devices to process location or direction information have recently started appearing. The popularity of these web services is due in part to the spread of cell phones and other mobile devices that come with functions such as GPS and accelerometers as standard features. The spread of Android and iPhone models has enabled development of systems and services that use sensor information, and a variety of applications have been released.

1.2 Reality-oriented web services
The paper refers to web services that use information obtained from mobile device sensors (such as location or direction information) as reality-oriented web services. A typical example of these services is the Sekai Camera[1] (‘World Camera’) service for the Apple iPhone. Released in November 2008 by Tonchidot, Sekai Camera displays information in camera images using iPhone features such as its GPS, accelerometer and magnetometer. Text, pictures and sound can be added to the camera images of mobile devices, a feature promising for use in new communication services displaying navigation information or advertising. So by using sensors to incorporate various real-world information, reality-oriented web services enable new features and modes of communication.

No development model exists for the existing reality-oriented web services that have been implemented so far. Since each has been implemented with its own unique implementing, developers have had to spend a lot of time and effort designing unmodeled areas, making high development costs a problem. The increasing number of reality-oriented web services being developed call for a way to lower development costs. This paper therefore proposes and investigates a method of modeling reality-oriented web services and a framework able to reduce development costs.

2. Current problems
This section presents the typical web application framework and VCCM (View-Controller-Controller Model) framework[4], the model for each, and the problems associated with developing today’s reality-oriented web services using these models.

Figure 1 illustrates a typical web application framework model. Figure 2 illustrates the VCCM framework model. Each framework is implemented using MVC (Model-View-Controller) architecture[2], a technique or structure that splits the software into ‘model’, ‘view’ and ‘controller’ elements for design and implementing. (Note that the model proposed by this paper and the MVC architecture model are two different models.)
2.1 Typical web application frameworks
Typical web application frameworks are widely used by developers as a way to reduce web service development costs. Ruby on Rails[3] is one of the most typical examples of a web application framework. It lets web service developers develop web applications just by placing the code as defined by the models set in the framework. Developers who use web application frameworks don't need to design new application configurations, greatly increasing web service development productivity.

![Figure 1 Typical web application framework model](image)

Figure 1 illustrates the model for developing a web service using a typical web application framework. The application pictured is a program that sends requests to the web service. The applications that typical web application frameworks are designed for are web browsers.

Typical web application frameworks support web services of the type that receive HTTP requests from client applications, process server data, and return HTML or XML. When developing this type of web service, developers don't need to write complex client data processes, so the web application framework model shown in Figure 1 doesn't model client processes. It therefore has the drawback of not being able to support development of reality-oriented web services that need to code for complex client processes.

2.2 VCCM frameworks
VCCM frameworks are designed more for coding client-side processes than typical web service development frameworks. They are specialized frameworks for developing web services used from programs that can code Flash and other rich processes in applications. Since VCCM frameworks support clients that can code rich processes, the controller used in MVC architecture has been divided into a client controller and server controller. In a VCCM framework, communication between server and client is defined by XML, so developers don't need any awareness of the communication processes. Figure 2 illustrates the model for developing a web service using a VCCM framework.

![Figure 2 VCCM framework model](image)

Using a VCCM framework lets developers write complex client processes, but since this model is not designed to enable the server to acquire data from the client, using it to develop reality-oriented web services that need to acquire data from sensors would be difficult.

2.3 Current development environment for reality-oriented web services
Reality-oriented web services need to acquire data from sensors. However, no existing framework defines a model for acquiring data from the client, and there is no concept for acquiring data from sensors. Moreover, no data format has been defined for transmitting data from the client, and to transmit sensor information from the client or transfer it to the view, the controller needs an integrated way to write the information. To develop a reality-oriented web service using VCCM for example, the developers would have to design the data format used for information transmitted from sensors, the sensor access method, and the data process integration method. For this reason, existing frameworks can't be used to develop reality-oriented web services.

3. Proposed model
3.1 Requirements for reality-oriented web services
To model reality-oriented web services, the following three elements must be designed in the client:
Data format used for transmitting sensor information: The format of the sensor information sent between client and server

Sensor access method: How data is acquired from the mobile device’s sensors

Data process integration method: The method of coding the flow of a series of data for processes such as acquiring data, communicating with the server, and passing data to the screen output application.

For example, a service could implement an application like Sekai Camera that displays information on the mobile device screen based on the mobile device’s current location information and the direction the camera is pointing. The data format used to transmit sensor information needs to define the format used for location information and direction information. To acquire this information, the application would need to acquire data from the GPS, accelerometer, and magnetometer. It would also need to code processes such as acquiring nearby-area information from the web server, and the method of mapping information onto the camera picture. The three elements above also make it simple to develop other applications besides Sekai Camera using the proposed model. These three elements are not supported by existing frameworks. The framework we have proposed aims to model these three elements which currently must be implemented independently when developing reality-oriented web services.

3.2 Model of proposed framework

To model the three elements listed in Section 3.1, this paper proposes a framework with MVC architecture in both the server and client. We have defined the format for transmitted data by defining a common sensor information format in the MVC architecture model of both client and server. Figure 3 shows the model of the proposed framework. The pictured sensor unit is the group of sensors in the mobile device.

There are two differences between the proposed framework and existing frameworks:
The proposed framework (1) enables the client to acquire sensor information, and (2) incorporates the format of transmitted data in the framework. When developing reality-oriented web services, these differences enable developers to code processes for handling various types of client sensor information in the framework. The proposed framework eliminates the need to design a new application configuration for the three elements described in Section 3.1, lowering development costs. The role of each element in the framework is described below.

[Client]
- Model: Defines the format of transmitted sensor information, and abstracts access to the supported sensors.
- View: Creates displays on the display screen, and sends events to the controller.
- Controller: Communicates with the server, acquires sensor information from the model, and transfers data to the view.

[Server]
- Model: Connects to the web server’s database, and defines the sensor information format.
- Controller: Communicates with the client, and acquires data from the model.

The most characteristic aspect of the proposed framework is the existence of a model element in the client. This model abstracts access to the sensors. Developers don’t access the sensors directly—they access them and acquire data via the model. The model also defines the format of transmitted sensor information, enabling it to handle all sensor-related processes. When developing reality-oriented web services, this approach enables the proposed framework to support elements that existing frameworks can’t support.

4. Framework design

This section describes an example design for
the proposed framework described in Section 3.2. Figure 4 shows the proposed framework’s internal structure.

As shown in Figure 4, the framework can be divided into five blocks—the ‘mobile device’, ‘model-view’, and ‘controller’ blocks for the client, and the ‘controller’ and ‘model’ blocks for the server. The mobile device is the program in the mobile device. It accesses the sensors and outputs information to the screen. The model-view abstracts the transactions with the mobile device element performed in the proposed framework. By programming the model-view, developers can develop services without programming the mobile device directly. The client and server controllers code processes pertaining to each data flow. The server’s model element also contains the same Protocol function, maintaining consistency in sensor information handling.

- Fetch Data From Sensor: A function that accesses the sensors to acquire data. Contains objects that abstract access to the sensors supported by the mobile device. For example, the current location information of the mobile device can be acquired by executing a GPS communication process that sends a request to acquire longitude/latitude or similar location information to an abstract object in the framework. Instead of accessing the sensors directly, developers acquire data by sending requests to abstract objects.

Controller:
- HTTP Connect: Communicates with the server’s controller using HTTP. Can convert sensor information to XML to send and receive it. Can also transmit information besides sensor information.
- Catch Event: Receives requests from the view. The client’s controller operates by receiving requests from the view. For example, a specified event is executed when a view button specifies an event defined in the controller and there has been a request for the button.

View:
- Display: Encodes processes pertaining to display on the display screen. For example, uses a button placement request to display the buttons on the display screen.
- Send Event: Receives the event that occurred on the display screen, and sends it to the Controller. Events are requests generated when a certain action has occurred (such as a button being clicked). When a certain event occurs, the action that will be called by the client’s controller can be specified. For example, pressing a button displayed on the screen calls the specified event.
Model:
- Protocol: Same as the client’s Protocol function.
- Fetch From Database: A function that accesses the database to acquire or update data. Contains objects that abstract communication with the database. This function has the same concept as the model element of existing web application frameworks such as Ruby on Rails.

Controller:
- HTTP Connect: Same as the client’s HTTP Connect function.
- Catch Action: Receives requests from the client’s controller. When the client’s controller specifies a certain action and sends a request, Catch Action calls the action defined by the server’s controller. Parameters can be acquired with HTTP POST and GET requests. XML data for sensor information can also be received as parameters.

5. Implementation
Using the proposed framework will enable flexible and low-cost development of various reality-oriented web services. As an example, we will consider the development of a sample application with functions similar to the Sekai Camera service described in Section 3.1. The requirements that need to be designed include:
1. the definition of the format of the mobile device’s location information and direction information,
2. the method of acquiring information from the GPS, accelerometer and magnetometer,
3. the method of acquiring nearby-area information from the web server,
4. the method of mapping information onto the camera image.

Figures 5 and 6 show the simulated Ruby on Rails code used to implement the controller processes of acquiring the mobile device’s location information and nearby-area information. Figure 5 shows the code for implementing the processes in the server’s controller, and Figure 6 shows the code for the client’s controller.

We will not show the coding for the view and model. The server’s controller starts processing when it receives a request from the client’s controller. This sample application operates by having the client send a request (containing the location information in a parameter) to the nearbyInfo action in the server’s MyServerController element, to acquire the nearby-area information for the location information passed in the parameter. The simulated code contains the LocationModel class and the InformationModel class. The LocationModel class is the model that manages location information. The InformationModel class contains the findNearbyInfo method that returns nearby-area data in response to the location information provided as the parameter.

LocationModel can convert back-and-forth between Location class instances that express...
location information, and XML for data communication. This ability enables it to convert to XML for data transmission, and to restore acquired parameters to Location class instances after acquiring them. XML conversion of location information is defined in the model element, enabling developers to write the code handling data transmission without worrying about data format.

The client also has a LocationModel class. Unlike the server, it has a toXML method that converts to XML, along with a getLocation method that acquires location information by accessing the GPS. Location information acquired from the getLocation method takes the form of Location class instances. To acquire location information using LocationModel, developers don’t need to worry about connecting to the sensors.

When developed using the proposed framework, it takes only about four lines of code each for the client and server to implement a program that acquires the mobile device’s location information and acquires nearby-area information. The proposed framework can therefore be used to greatly reduce development cost.

6. Conclusion
As mobile devices with sensors become more widespread, the development of reality-oriented web services that process sensor information should become an increasingly important area. In response to this trend, this paper has proposed a framework that addresses the problems of existing frameworks and simplifies the development of reality-oriented web services. To model the approach used to process sensor information, we have designed a framework with MVC architecture in both the server and client. We have also created a configuration that defines the data format used for sensor information transmission within the framework.

The proposed framework enables reality-oriented web services to be developed without the developers having to be aware of the data format used when accessing the sensors or during data transmission. Using the proposed framework can reduce costs when developing reality-oriented web services.

Issues for the future investigation include implementing designed frameworks to evaluate their run-time overhead, and using them for actual applications.

References