DESIGN STRATEGIES FOR WEB BASED ITS APPLICATIONS
A Proposed Architecture in Design of Intelligent Transport Systems Application

Andrea Sponziello
University of Salento, Lecce, Italy

Keywords: Mobile, scripting, web services, ITS, transportation.

Abstract: Modern ITS (Intelligent Transportation System) applications are designed on Web paradigm, exploiting third-party web services for accessing, for instance, to geographic information and routine such as cartography, route planning or weather condition services. This allows a customer to use the ITS platform as an outsourcing service, due to the Web interface, reducing maintenance costs and improving the application accessibility.The web technology creates only a basis for an easy SOA (Service Oriented Architecture) integration. In order to have a full SOA compliant ITS application we have to adopt a new perspective for the ITS applications design: from a monolithic and closed web application to open framework of services and APIs capable of user customizations.In fact, a third-party application like an ERP (Enterprise Resource Planning) can take many advantages by linking its vehicles, drivers, workers (and mobile resources in general) directly with real-time and historical data of the ITS. In this paper we demonstrate that integrating an ITS application developed on the Java platform with modern Scripting Engines and Languages such as Groovy, JRuby or Mozilla Rhino greatly enhances the possibility of extending and integrating the application itself with external applications.

1 INTRODUCTION

This study concerned the specific field of developing Mobile Fleets Management (MFM) applications used for monitoring a large number of mobile resources on the territory using GPS devices that transmits their positions with Over The Air transmission protocols as GPRS/UMTS to a Data Center.

This programmable devices are mounted on Mobile resources as Cars or Trucks and transmits GPS position and other stuff to a Data Center that stores this informations for later use. The registered data is used to show real-time position of resources on Maps, generate reporting for data analysis or generate alarms on events based on geographic position of the resources (*geofencing*).

The aim of this work is to provide an ITS web application with some extension points. An extension point is a part of the application where the user can introduce customized code, changing the application behaviours in order to meet his needs.
In the context of this work we call *User* a Trusted Developer of our application, Known by the Application Administrator and able to plug the application with new features, applying the methodology that we describe in the following.
We worked on an existing ITS application called Realtrack (www.realtrack.it) to introduce some of the extension points that are the subjects of this article. This application is completely developed in Java language using many Open Source frameworks like Jakarta Struts, JBoss, Postgres/PostGis and other OS frameworks. We developed extensions applying many refactorings to the application architetcture in order to meet the new proposed specifications.

2 DESIGN OF A WEB BASED ITS APPLICATION

Here follows the main new specifications we considered in the refactoring process of the web based ITS application.

1. The application can publish a set of predefined *events* (extension points) that programmers can use to write their custom actions when an event occurs. This set of

events is available in many parts of the application, giving user the possibility to highly customize the application basic functionalities. We call this module *Event Handling Module* (EHM)

2. The final user can write his own web services to be embedded directly into the application. In this way the application can be easily extended with the introduction of new services that can be served to some external applications supplying some extra functionality (ex. ERP applications). We call this module *Web Services Extension Module* (WSEM)
3. Allowing a sort of dynamic class-loading (as in Java Language) we can give to the user the possibility to plug/unplug parts of the application modifying the standard application behaviours simply by changing in a text file (XML or properties file) a class-name declaration (the extension point) with another one. This can be easily obtained using a Dependency Injection Framework in Application Design like the *Spring Framework* or the *Jboss Microcontainer* if the application is written in the Java language. We call this module the *Application Microcontainer*
4. The application can be logically divided into many services that can be plugged/unplugged based on business needs. For example the graphical user interface (GUI) can be decoupled by the low level services so that customers can use only low level services to write their own, customized, GUI. If the application is designed as an aggregation of services also the API framework used for writing the application can be published to final users that will use the APIs to interoperate with the various services. For example a specific service of the application can act himself like a provider and can publish a set of client-API that can be used by the GUI framework to communicate with this provider taking the information needed to render a web page. In this case the entire Service (like the GUI module) can be considered an extension point because of the possibility to plug it with a completely new module written by the final user that unplugs the original one shipped with the application.

For the purpose of simplifying customized code writing we built a Scripting Engine called SEE (Scripting Embedded Engine) (Sponziello, 2006). It is behind the scenes of many modules and gives users the possibility to write custom code in the easy way provided by scripting languages instead of compiled ones (like Java). The primary script languages used by our framework are basically Groovy, JRuby and Javascript (see References). They are directly supported and well integrated by the Java platform. This script technologies easily provide a simple way to write custom event handlers, routines or Web Services without the need of recompilation and with a more user-friendly syntax. It is important to mention the MOP (Meta Object Programming) capabilities of the Groovy and JRuby languages that helps programmers to easily write DSL (Domain Specific Languages) or the open classes and duck typing features of either Groovy, JRuby and Javascript that enhance flexibility of code scripts giving some features unavailable to the Java language that we used to write the application itself (Freeman, 2006).

2.1 The Event Handling Module (*EHM*)

The application we are considering supports a wide set of events, such as:

Table 1: Main application Events.

Event Group	Events
Communication	*OnNewPosition*
CRUD	*OnMobileNew, OnMobileUpdate, OnMobileDelete*
GIS	*OnMapDraw, OnLayerDraw*
GUI	*OnLogin, OnPageRenderStart, OnMenuRender*

The application processes the arrival of a new GPS position from the mobile resource in this way:

- stores the position into the database (latitude, longitude, speed, direction, altitude, board signals, sampled weather conditions etc.)
- updates related position information (vehicle status for example)
- verifies some *geofence* (TODO: explain Geofence) feature and eventually notifies that event occurred with a standard message (sms and/or email) to a set of user-defined addresses.

Some of the data used by this process was previously defined by some user with the visual interface of the application. This user-interface gives a way for visual creation of Geofences with related informations about SMS and/or email notifications.

2.1.1 Use Case 1: Custom Geofences

Now suppose that a user would change this standard behaviour adding or extending the actions that belongs to the event of a new position arrival (*OnNewPosition* event). The framework accomplishes this need providing the user with the possibility to add a custom script that implements the desired actions.

A simple Use Case illustrating the standard application support for geofencing would be:

1 - the user graphically defines a geofence as a circle geometry specifying a center and a radius using the standard application GUI

2 - the system notifies via sms and/or email a vehicle entering the geofence, communicating this event with a standard message.

The user wants to extend the standard action related to the previous event, sending a notification only if this particular conditions occurs:

- the mobile resource (a vehicle i.e) entered or exited the geofence (standard - provided by application)
- the vehicle is over a maximum speed (i.e. 80 km/h) (user defined - not provided by the application)
- the weather condition is "raining" (user defined – not provided by the application)

This type of advanced notification is not provided by the application standard framework that offers only notifications about entering/exiting from the geofence's boundary.

The user can so write his own script to handle the customized event. He selects from the events' list the preferred event to handle, in this case onNewPosition. Then he writes and uploads a simple text file containing the script. This script is then registered by the System and will be executed when every onNewPosition event occurs.

The event onNewPosition arises when a new position is communicated by the vehicle's device.

For example the following script could be written to handle this Use Case:

```
onNewPosition:
  // the position built by the
framework
  position = protContext.curr_position
  weather =
service.ws.call("http://www.weatherserv
ice.com/ws", position.lat,
position.lon, new Date() )
  geofence1 =
protContext.curr_geofences.my_geofence
  if (geofence1 &&
             position.speed > "80kmh"
             &&
          weather.isRaining() ) {
    geofence1.disable()
    // removes standard actions
    geofence1.removeSmsNotifications()

geofence1.removeEmailNotifications()
    // a custom message to the driver's
mobile phone

geofence1.addSMSNotification("+39329352
9381", "DECREASE SPEED, PLEASE!")
    // a custom message on the user
display

protocolContext.response.add("USER_MESS
AGE","DECREASE SPEED, IT'S RAINING!")
  }
}
```

Script 1: Custom Geofences.

The application scripting framework provides the injection of some context objects inside every event handler script to allow the script author to use the available system resources.

For example the injected object protocolContext publishes a set of useful environment properties as the current generated position (protocolContext.current_position) or the current active geofences on the mobile resource with protocolContext.currentGeofences.

Another useful property published by protocolContext is protocolContext.vars that is a map of all raw variables arrived from the vehicle device. Using these variables a user can adjust coordinates in order to prevent systematic GPS errors before storing the position into the database.

Another useful property available in the script is protocolContext.response that can be used (as in script 1) to modify the response sent to the client device.

Other objects are injected by the framework inside the script. For example the injected service object that can be used to call remote web services (service.ws, as in script 1)

2.1.2 Use Case 2: Custom Data Warehouse Creation

Another useful customization could be the creation of an alternate positions' datawarehouse (i.e. for a business intelligence integration) in a location different from the ITS database. An example script follows illustrating this possibility:

```
onNewPosition:
    position =
protocolContext.current_position // the
position built by the framework
    // invokes a remote WS to perform
geo-data ingestion to an external
datawarehouse

service.ws.call("http://wharehousesite.
com/ws",position.lat, position.lon,
           position.speed)
```

Script 2: Custom Data Warehouse creation.

2.1.3 Use Case 3: Custom Notifications

Another task that can be accomplished with the application scripting extensions are the notification of the changed status of a vehicle as in the following script:

```
onNewPosition:
    previous_pos = position.mobile.pos2
// MOP method
    if (previous_pos.status ==
position.ENGINE_OFF ||
            position.status ==
position.ENGINE_ON ) {

service.ws.call("http://ws.site.org/ws"
, …)
    }
```

Script 3: Custom notifications.

2.1.4 Script's User Profiling

Every script can be saved on a "per user" profile and can also be called in chain mode (multiple scripts per event) so that every event's script can manage specific tasks and prepare the environment for the next script.
In this way every application user can have its own scripts handling its own tasks. For example the administrator can write some scripts enhancing some kind of events that are executed before others. Less privileged users can write their script that are executed after the administrator's one. Naturally, the application administrator can disable scripting for some users.

An example of a script chain is the following:

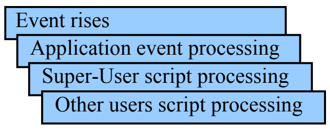
Event rises
Application event processing
Super-User script processing
Other users script processing

Figure 1: Script execution chain.

2.2 WSEM (Web Services Extension Module)

The WSEM module allows the user to write and publish custom Web Services directly embedded into the platform. In this way the standard application can be extended with the introduction of new services that external applications can call to retrive particular informations. Using this module we can see the application itself as a service provider for other applications and permits easy SOA integration. To allow the user writing custom web service we used the REST (Representational State Transfer) methodology that simplifies the writing of the Web Service letting the user to publish it using a customized protocol (not SOAP) and a very simple calling method. The REST approach really simplifies the calling of a remote service relying on the classical HTTP way instead of XML over HTTP typical of the SOAP approach.

This permits to a Web Service to return not only XML but also other kind of data as *JSON* (very useful for AJAX calls) or Javascript, *HTML* (useful to plug new GUI components) , *plain text*, *YAML*, *pdf* etc.

To give the user the possibility for easily create a Web Service we used a MVC architecture for the WSEM module. In this architecture the *actions* and *views* are stored inside the application DB in a virtual File System instead of the real one so the services can be created and tested directly from inside the application itself in a multiuser fashion.

Every action is composed by a Controller component and one ore more View components. As for the EHM module some context objects are automatically injected by the application inside every controller action, and some of this objects represents the Model component of the MVC Pattern.
The controller scripts are written in the Groovy (**reference**) language while the View components are written in the Velocity (see References) language. To realize this module we developed an

engine called JControl that we published on sourceforge web site, (see References). The JControl way of creating new end-user webservices is very simple. The User that has permissions to create new web services can enter a special zone of the web application that acts as a mini IDE for writing web services. The user have to choose a url route for the new web service and then create a new Controller component. This component is simply a Groovy script that the framework will execute when the url is called with an HTTP request. Into the script, as for the case of the **EHM** module, some objects are created directly by the framework. This objects are used to communicate with the Model, use some APIs to call external web services, save files on the virtual File System etc. Here follows some use cases that illustrate this kind of functionalities.

2.2.1 Use case 1: WS to get Mobile Resources of a Particular Fleet (XML and HTML)

As we said before every web service is built with 2 components: a controller component (the logic) and a view component (the presentation). A web service can present data in text format suitable to be consumed by a machine (XML, JSON etc.) or by a Web Browser (HTML). In the second case the web service module can be used to build new GUI components for the application. Before writing the new Web Service the User must choose a new Url for the web service itself. In the first Use Case the user wrote a web service to get all the Mobile Resources that belongs to a particular Fleet. This web service will be called with an url like this: *http://site/custom/get_fleet_mobiles?id=12*

URL: /custom/get_fleet_mobiles

Controller code (Groovy language):
```
fleet_id = http_request["id"]
fleet = FleetManager.find(fleet_id)
mobiles = fleet.all_mobiles()
view.put("fleet", fleet)
view.put("mobiles", mobiles)
```

View code (Velocity language):
```
#if ($type=="xml")
$request.setContentType("text/xml")
<xml version="1.0">
<document>
  <fleet>
    <name>$fleet.name</name>
    <mobiles>$mobiles.size</mobiles>
  </fleet>
#foreach ($mobile in $mobiles)
  <mobile>
    <name>$mobile.name</name>
    <type>$mobile.type</type>
    <id>$mobile.id</id>
    <position>
      <lat>$mobile.pos.lat</lat>
      <lon>$mobile.pos.lon</lon>
      <time>$mobile.pos.ts</time>
      <speed>$mobile.pos.speed</speed>
      <dir>$mobile.pos.direction</dir>

<weather>$mobile.pos.weather</weather>
    </position>
  </mobile>
#end
</document>
#else
$request.setContentType("text/html")
  ## similar but HTML instead of XML
  ## ...
#end
```

2.2.2 Use case 2: Get Last N GPS Data for a Particular Mobile Resource in KML Format

KML (Keyhole Markup Language) is the XML used by *Google Earth* and *Google Maps* Applications to display external GIS data. This Use case shows how to use the WSEM to generate this kind of data.

An example URL for this Use case is: *http://site/custom/get_locations?id=67&pos_num=50&type=kml*

URL: /custom/get_locations

Controller code (Groovy language):
```
mobile_id = http_request["id"]
n = http_request["pos_num"]
mobile = MobileManager.find(mobile_id)
// last_pos is an array of N elements
last_pos = mobile.las_pos(n)
view.put("mobile", mobile)
view.put("last_pos", last_pos)
```

View code (Velocity language):
```
#if ($type=="kml")
$request.setContentType("application/vn
d.google-earth.kml+xml")
<?xml version="1.0" encoding="UTF-8"?>
<kml
xmlns="http://earth.google.com/kml/2.0"
>
 <Document>
#foreach ($mobile in $mobiles)
  <Placemark>
   <name>$mobile.name</name>
   <description>
```

```
    <![CDATA[$mobile.description]]>
   </description>
   <Point>
    <coordinates>
      $mobile.position.lat,
      $mobile.position.lon
    </coordinates>
   </Point>
  </Placemark>
#end
 </Document>
#end
</kml>
#else
  ## other types supported
  ## ...
#end
```

3 SECURITY ISSUES

Particular attention must be accomplished regarding securing the User Script Extensions. Not every User can use the same System Resources like reading or writing Files on local Filesystem, Opening arbitrary sockets on arbitrary machines over the Internet, Writing arbitrary data into the Database. Because of the java Technology, we can use, indipendently of the language (Groovy, JRuby or Javascript), the Java SecurityManager Technology to perform some actions that prevents some users to access some resources. The reliability of this approach (using only the SecurityManager) has not still greatly covered by our study but we reserve to do it later. By now this is not a great problem because the user base is not so wide, but the growing number of users impose to consider the Security
a primary problem.

4 CONCLUSIONS

The proposed extensions give to the final users the possibility to develop custom code to modify system's behaviours without the knowledge of the application source code, language or other complex low level design details.

Moreover application's developers can delegate to final users the addition of particular features. There is no need to create and maintain new software branches for user-specific customizations, that is a very costly and complex work. The Application's designers will focus on the development of the main framework with integrated customization capabilities due to EHM and WSEM integration. If a customer needs some customization a developer can solve the problem simply writing some macro without modifying the source code of the application creating a new branch for this customer that needs to be maintained.

Because the application we used for experimenting with the proposed extensions is a Web application written with Java language, the scripting technologies used for this work are Java based scripting engines in particular Groovy JRuby and Rhino. These are not the only Java supported scripting languages but are the best suited for our purposes because of their standardization inside Java technology.

The users can write script *macros* that will be executed in particular points of the application's computational lifecycle. We call these *extension points* and have the form of *events* that raise in particular lifecycle steps. For example, the arrival to the ITS server of a new position from the moving vehicle might trigger an event named *onPositionNew*. The user who wants to override the standard application behaviour corresponding to the arrival of a new position can write a macro that will be executed by the system every time this event occurs.

With macros the user can, for example, modify the geographical coordinates before they are saved to the Data Base or he can send the coordinates to a third-party remote application in order to enrich a Data Warehouse and to perform advanced Business Intelligence features. Moreover the user can generate custom notifications to particular users entering particular geofences. The use of the WSEM module allows easy publishing of application's data and functionalities to external applications.

ACKNOWLEDGEMENTS

This work has been developed in the REALTRACK industrial research project in collaboration within the Center for Advanced Computational Technologies of the University of Lecce (Italy) directed by the Prof. Giovanni Aloisio. The project was partially funded by the Regione Puglia (Italy) under the POR 2000-2006 measure 3.13. For further information, please refer to the Web page http://www.realtrack.it.

REFERENCES

Freeman S., 2006. Embedded Domain Specific Language. In *OOPSLA'06,* http://www.jmock.org/oopsla2006

Fernandez O., 2005. Working on the Rails Road, In *Object Technology User Group Meeting, Minneapolis, 2005*

Sponziello A., 2006. Scripting Support Extensions for ITS Applications, In *FOSS4G'06*, http://www.foss4g2006.org/contributionDisplay.py?contribId=174&amp;sessionId=42&amp;confId=1

Fensel D.,

Jcontrol, An MVC Framework for the Java language, http://sourceforge.net/projects/jcontrol/

Groovy, A Scripting language for the java paltform, http://groovy.codehaus.org/

Jruby, A Ruby implementation for the java platform, http://jruby.codehaus.org/

Rhino, A Javascript implementation for the Java platform, http://www.mozilla.org/rhino/

Velocity, A text templating engine, http://velocity.apache.org

Cowan, Lucena, March 1995, "Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse", IEEE Transactions on Software Engineering, Vol 21 No 3.

Krasner, Pope, 1988, "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80", Journal of Object Oriented Programming, August/September.

Parr, Terence (2004) Enforcing Strict Model-View Separation in Template Engines. In *Proceedings International WWW Conference*, New York, USA. *http://wwwconf.ecs.soton.ac.uk/archive/00000578/01/p224-parr.pdf*

Krasner, G., Pope, S. (1988) A description of the model-view-controller user interface paradigm in the smalltalk-80 system. Journal of Object Oriented Programming, issue 3, volume 1.