CAMEL FRAMEWORK
A Framework for Realizing Complete Separation of Developer's and Designer's Work in Rich Internet Application

Hiroaki Fukuda and Yoshikazu Yamamoto
Graduate School of Science and Techinology, Keio University
3-14-1, Kohoku-ku, Hiyoshi, Yokohama Kanagawa 223-8522, Japan

Keywords: Framework, Rich Internet Application, Web Engineering, Software Engineering, Flex, Web Application.

Abstract: This paper provides a framework called Camel which is able to separate developer's and designer's work completely in developing Rich Internet Applications. In the last years, web becomes one of the most popular methods to transfer information such as search contents, advertise information and online shopping. We usually use an appropriate web application to achieve our requirement. In the development process of current web applications, designers design interfaces of an application by using HTML, CSS, image and developers not only implement business logics but also modify the interface for dynamic generation of HTML tags based on the executed logic. Therefore, not only designers but also developers have to work every time when design of the interface is changed.
This paper describes camel framework and its implementation, which is based on Flex, and then demonstrate its usage and utility by implementing a real application.

1 INTRODUCTION

Web has become not only basic infrastructure to open and refer information but also development platform of all sorts of integrated business solutions. In the last years, a large number of web applications exist on the internet. Due to the increasing complexity of these application, current web technologies are starting to show usability and interactivity limits. The user experience in thin-client web applications is not comparable to desktop applications, responsiveness is lower due to network overhead and unnecessary round-trip server access, and disconnected usage is not supported (Driver et al., 2005).

Rich Internet Application (RIA) has been recently proposed in order to solve the problems of current web applications described above (Duhl, 2003). They provide sophisticated interfaces for representing complex processes and data, while minimizing client-server data transfer.

On the other hand, when we think about the development process of current web applications, designers design all interfaces by using HTML, CSS and attractive objects (e.g., image, swf, movie). Meanwhile, developers implements server-side logics that are invoked base on the requests from clients by server-side technologies such as Java, PHP, ASP. In this process, they have to modify codes of interfaces created by designers in order to include logics for validation and/or dynamic creation of HTML tags as a result of server-side logics. Therefore, if a requirement to modify the design of interface happens, not only designers but also developers have to work for the modification, testing, and re-deployment.

This paper proposes a RIA framework, called Camel, which can realize complete separation of developer's and designer's work in development processes. It uses and extends Flex (Adobe Systems Inc., 2007b), which is one of the RIA development tools, and can separate client-side logics from interface source codes, so that developers and designers do not need share any source codes in an application. In Flex, we basically have to specify event handlers as an attribute of each interface component directly in order to execute logics correspond to events that are dispatched by user's operations. However, Camel injects logics into interface components dynamically at runtime so that it can separate logics from interface components completely by source code level. By using Camel framework, developers do not care about the design of an application and can concentrate on logic implementation.

This paper is organized as follows. We describe the background and overview of RIA technologies in

Fukuda H. and Yamamoto Y. (2008).
CAMEL FRAMEWORK - A Framework for Realizing Complete Separation of Developer's and Designer's Work in Rich Internet Application.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - ISAS*, pages 137-143
DOI: 10.5220/0001684401370143

section 2. We then describe basic ideas of this framework and related technologies in section 3. In addition, we explain the implementation of runtime system and basic rules to use this framework in section 4 and application behavior in section 5. In section 6, We also show a real application implemented by using this framework to show the usability and then describe related work in section 7. In section 8, we conclude by providing summary and discussing future issues.

2 BACKGROUND

This section briefly describes the background of this framework.

2.1 Development Process

Web applications have become important tools for companies to advertise themselves. In addition to the usability of an application, interface design, including base color, location of each component (e.g., images, swfs, movies), becomes important because it influences user's mind. Therefore in case of developing a web application for consumers, designers design entire interfaces and then developers implement logics. We briefly explain this process in Fig.1

1. Designers design each page as an image by using appropriate tools (e.g., photoshop and illustrator).
2. Clients who order the website confirm entire design and ask designers to modify some parts.
3. Once the entire design is fixed, designers (or HTML coders) divide the image based design into each image object (e.g., titles, buttons and icons) and then create HTML based interfaces.
4. Developers implement business logics.They have to modify HTML based interface created by designers to generate HTML tags dynamically.

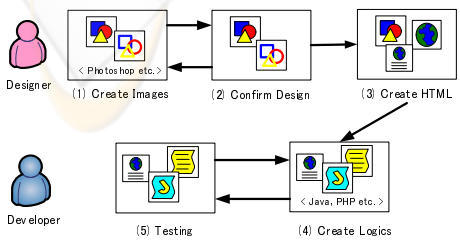

Figure 1: Development process of current web applications.

5. Developers make unit and/or integrated tests of an applications. If they find bugs in this process, they go back to step 4 and fix them.

Current web applications are implemented based on MVC architecture, therefore, server-side logics are independent from interfaces. However, developers need to include program codes in HTML based interfaces for generating appropriate HTML tags based on the Model. This is a quite difficult process for developers because HTML based interfaces created by designers are difficult to understand. Moreover, if the clients require to modify the design after step 3, designers and developers have to redo from step 2 to step 5. This takes quite a little cost and time, however, the change of design or specification during development processes is usually happened because the client can not image usability (including design) of the application from the beginning. In the development process of current web applications, it is impossible to separate programs from interfaces completely. Therefore, this limitation forces to share interface codes between designers and developers.

2.2 Overview of RIA Technology

The term "Rich Internet Application" was introduced in a Macromedia whitepaper(Allaire, 2002). Several technologies have been proposed to support RIA development and they can be classified into four categories as shown below

- Scripting based
  The client side logic is implemented by scripting language such as JavaScript and interfaces are based on traditional HTML and CSS.
- Plug-in based
  Advanced rendering and event processing engine should be installed as a browser's plug-in (e.g., Flash,(Adobe Systems Inc., 2007a) Flex, Laszlo(Laszlo Systems Inc., 2007)).
- Browser based
  Rich interaction is natively supported by some browser that interpret declarative interface definition language (e.g., XUL(Mozilla.org, 2007)).
- Web based desktop technology
  Applications are downloaded from the web, however, executed outside the browser (e.g., Java Web Start(Sun Microsystems Inc., 2007), Smart Client(Microsoft Inc., 2007)).

2.3 Flex Framework

Flex is a framework released by Adobe Systems for the development and deployment of cross platform

```
<mx:Button label="test" click="Alert.show ( 'Hello' )"/>
```
(a)

```
< mx:Script >
    private function initialize () : void {
        id.addEventListener (MouseEvent.Click, hello );
    }

    private function hello () : void {
        Alert.show("Hello");
    }
</mx:Script>

< mx:Button label="test" id="testButton"/ >
```
(b)

Figure 2: Flex Programming.

Rich Internet Applications based on Adobe Flash platform. It has two languages named MXML and ActionScript3.0 (AS3). MXML is a XML based language that designers/developers use to define the interface of an application. In the deployment process of Flex applications, designers design interfaces with MXML and images. In addition, developers implement client-side logics with AS3 and server-side logics with server-side technologies.

3 APPROACH

This section briefly outlines the basic concept of Camel framework presented here.

3.1 Flex Programming Model

Applications created by Flex are working by event-driven architecture. When the user interacts with the interface, and also when important changes happen in the appearance or life cycle of a component, such as creation or destruction of a component or its resizing, appropriate events are dispatched and event handlers handle the events. In Flex, there are two ways to associate events and event handlers. First, in Fig.2(a), every MXML component owns its dispatchable events as attribute names and handler functions which will handle the events are defined as the values of attribute. Secondly in Fig.2(b), we use "addEventListener" method for these associations. Every MXML component has "id" attribute so that we can identify each component and we can refer every component by using value of "id". Therefore as shown in Fig.2(b), we can add a function as an event handler to a component in which the first argument represents event name and the second is the reference of function. In addition, all processes are working on Flash player in some browsers except RPC calls. When the user require further information/data, an application will get it from server via RPC components prepared in Flex. When these components complete their processes, they dispatch result/fault events, therefore, developers also add event handlers to handle the results.

3.2 Loosely Coupled Components

As we explained before, almost all web applications are constructed by MVC (Model View Control) architecture to maintain their scalability and adaptability. Therefore, Flex applications should be constructed by the same architecture. In MVC architecture, View will be changed based on the values of Model, that is, there is strong dependency between them. On the other hand, Control will dispatch appropriate business logics based on the type of received requests. We believe the dependency between logic and View should be weak because a business logic can be dispatched by several requests.

As we explained above, in traditional web applications, we can not separate View sources from logic sources completely because of the strong dependency between View and Model and there is no way to separate them in development and execution processes.

On the other hand, Camel makes it possible to separate View from other components e.g., Model and logic in development processes and join them in execution process introduced by dependency injection concept explained next section.

3.3 Dependency Injection

Dependency injection (DI) aims to solve a particular problem in designing and constructing data structures dependent on other pieces of code. For example in Fig.3(a), class A has codes to create class B and use methods of class B That is, class A can not be run without class B, in other words, class A depends on class B. On the other hand in Fig.3(b), class A has only codes to use methods of class B and the codes to create class B are included in DI Container. That is, class A does not care about the creation process of class B and DI Container injects the dependencies between class A and B at runtime. As a result, the dependency between class A and class B becomes weak.

In the context of this paper, the definition of event handler in MXML tags and "addEventListener" method in ActionScript make these dependency between View and Model.

4 IMPLEMENTATION

Camel makes use of tree structure of display objects and reflection mechanism in ActionScript. We de-

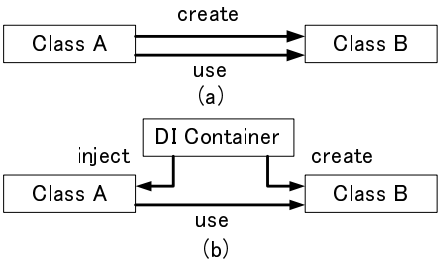

Figure 3: Dependency Injection.

scribe them first and show the structure of Camel runtime system. Secondly, we explain basic rules when we use this framework.

4.1 Display Object Tree

"Application" is a root object of a Flex application and every visible component, which must extend "UIComponent" class, is added. In addition, there are two ways to add an object to its parent. First, we can invoke "addChild" method of the parent object where the first argument is an added object. Secondly, we can specify not only visible component but also invisible component (e.g., Model or logic class) as MXML tags in MXML file. The nested structure of MXML file represents tree structure of Flex components. Therefore, we can get all references of added objects by traversing this tree.

4.2 Reflection Mechanism

Like Java and Ruby, AS3 has powerful reflection mechanism. In the reflection mechanism of AS3, an object is translated to a XML object which includes its properties, methods, parent classes etc. and we can access them by EX4(Ecma International, 2005) style. Therefore, it is easy to invoke a method by using only method name and its arguments at runtime if we can get the reference of the target object.

4.3 Runtime System

Fig.4 outlines the basic structure of a runtime system and its entire architecture. As we explain next section, developers have to prepare Listener class that has handler functions for dispatched events from visible component in View. A runtime system consists of five components as follows.

- **Component Manager** registers/removes the references of View and Model components. It also manages hierarchical composition of View, Listener and their parent. A Listener component should be located at the same level of the corresponded View, so that "DI Manager" can get the reference.

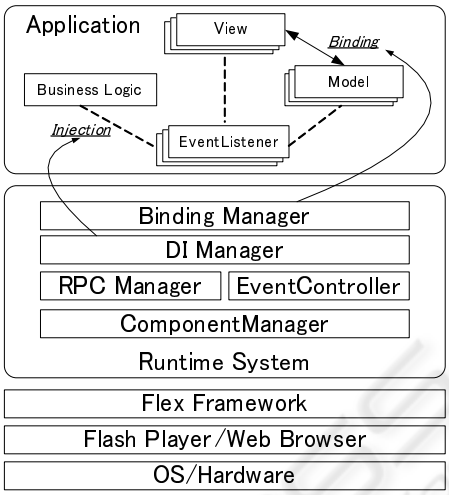

Figure 4: Runtime system and architecture.

- **EventController** invokes an appropriate business logic based on the dispatched event. This is not a Flex event but an application specific event defined by developers to invoke a business logic. Therefore, developers have to register logic classes and event types to the class which extends EventController.
- **RPC Manager** stores Flex RPC components such as HTTPService, WebService, RemoteObject, Producer and Consumer.
- **DI Manager** executes three types of dependency injection at runtime. First, it injects not only handler methods of Listener components into events that are dispatched from visible components in View but also the references of View and Model component into Listener component as members. Secondly, it injects logic methods in business logic components into application specific events mentioned above. Finally, it also injects callback methods in business logic component into the events which are dispatched from RPC components.
- **Binding Manager** binds values between visible components in View and members in Model. This binding is based on the name of visible components and members in Model. For example in Fig.5, a member "dataGrid" in Model will be bound to the visible component named "dataGrid". In addition, the type of member in Model such as String, Number, Date, etc. is predefined in Camel, e.g., String for TextInput, ArrayCollection for DataGrid etc.

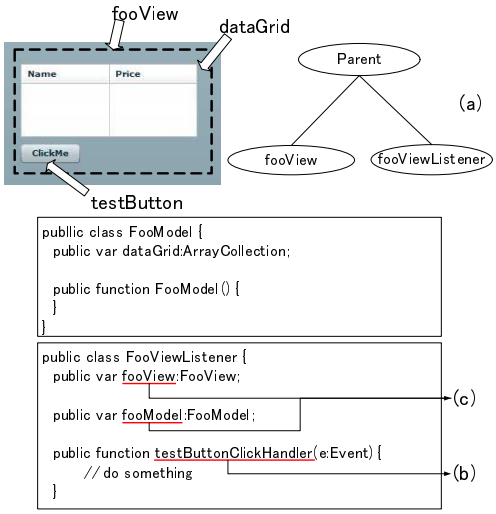

Figure 5: Relationship between View, Model and Listener.

```
<ServiceRepository>
  <mx:WebService id="myserv" url="http://aaa.bbb/"/>
</ServiceRepository>
```

```
public class FooLogic {
  public var myserv:WebService

  public function loginEventHandler(e:Event) {
      // do something
  }
  public function myserv_doLoginResultServiceHandler {
      // do success process
  }
  public function myserv_doLoginFaultServiceHandler {
     // do fault process
  }
}
```

Figure 6: Relationship between RPC component and Logic.

4.4 Basic Rules

Camel is inspired by convention over configuration concept introduced by Ruby on Rail (37signals, 2007). Therefore, it has several name based rules instead of XML formalized configuration files. Basically, we need four components (e.g., Model, View, Listener and Logic) to develop an application and one View corresponds to one Model and one Listener. Based on this, we explain these rules to support abovementioned injections and the binding as follows.

1. Every component defined as MXML tag must have id attribute and value.
2. A View and its corresponded Listener should have the same parent in the display object tree (Fig.5)(a).
3. View name must end "View" character and Listener name must end "Listener" character in addition to corresponding View name(e.g., fooView and fooViewListener) (Fig.5)(a).
4. The name of methods in Listener, which handles dispatched event from visible components in View, must be defined as "component id" + "event name" + "Handler" characters. For example in Fig.5(b), a button component named "testButton" dispatches "click" event and then we have to define the method to handle it as "testButtonClickHandler" in the Listener.
5. Listener must have the references of corresponding View and Model inside it. In addition, The reference name of View must be the same as View name described above (fooView) and also the reference of Model must be named as "fooModel" where the suffix is replaced with "Model" from View name Fig.5(c).
6. Model must have the reference of values that are bound to visible components in View. The reference type is based on the visible component type which is predefined by Camel as we mentioned section 4.3.
7. The name of methods in Logic, which handles application specific events dispatched by developers must be defined as "event name" + "EventHandler" characters. For example, a developer must define "loginEventHandler" method in Logic for handling "login" event. In addition, if we use RPC components in Logic, the Logic must have the reference of the RPC component and two types of methods (success or fault) to handle the result. For example in Fig.6, when we use a WebService component, we have to define the component as MXML tag with id as a child of ServiceRepository tag, and then define the reference in Logic where the name must be the same as the id in ServiceRepository. Then, we have to define callback methods as "component id" + "_" + "method name" + "Result/Fault" + "ServiceHandler". In Fig.6, "myserv_doLoginResultServiceHandler" and "myserv_doLoginFaultServiceHandler" are callback methods for "doLogin" service invoked by WebService component named "myserv".

5 APPLICATION BEHAVIOR

This section describes about application behavior including when and how the dependencies are injected

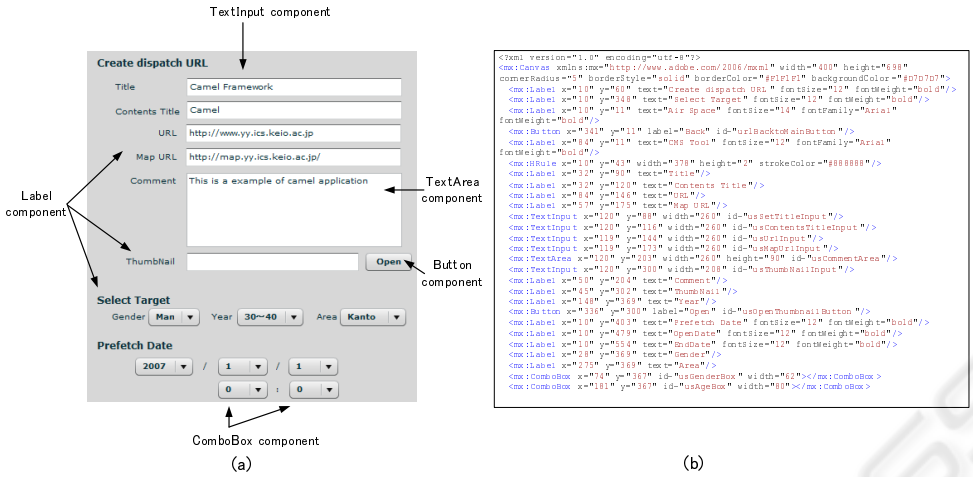

Figure 7: Screenshot of CMS and View code.

by using an application introduced in Fig.5 and 6. First of all, a developer has to use “CamelApplication” as a root tag instead of “Application”. Based on this, dependencies are injected as follows.

1. An application starts.
2. Camel catches all “creationcomplete” events of its child components by “CamelApplication” class. This event is dispatched by Flex when a component completes its initializating process.
3. Camel gets the references of FooView by component id (fooView). As we described section 4.4, the name of View must end “View” characters.
4. Camel also gets the reference of FooViewListener corresponding to the FooView by using component id and tree structures in Fig.5(a).
5. Camel parses XML of FooView and FooListener by using reflection mechanism of AS3 and injects “testButtonClickHandler” method in FooListener into “testButton” component in FooView based on the naming rules explained in section 4.4. In addition, Camel also injects the references of FooView and FooModel into fooView and fooModel in FooListener. If Model is not created, Camel creates and registers it to “ComponentManager” to keep each Model singleton.
6. Camel binds “dataGrid” in Model to dataGrid which is visible components in View based on the naming rule. This binding uses “BindingUtil” class in Flex.
7. When an application dispatches “login” event, Camel create FooLogic and parses it to inject an “loginEventHandler” method in FooLogic into “login” event. In addition, Camel injects the reference of WebService component named “myserv” into FooLogic and then injects “myserv_doLoginResultServiceHandler” and “myserv_doLoginFaultServiceHandler”methods into callback events dispatched by WebService component.

6 EXPERIMENT

We developed a real application which is a content management system by using Camel. It has 15 View and Listener components in addition to 5 Logic components. In addition, it has create, update, delete, search functions for management of data and can upload/download contents from/to servers. It uses RemoteObject as RPC components in Logic to serialize application specific data in database on a server.

As shown in Fig.7(a), View has several visible components such as TextInput, TextArea, Button, ComboBox, DataGrid and Label. On the other hand, as shown in Fig.7(b), there is no programming codes in View component. This means that Camel can realize complete separation we focus on and is adaptable for real applications.

7 RELATED WORK

There are various frameworks in current web applications. Zend framework (Zend Technologies, 2007) and Mojavi (Major Computing, 2007) are PHP based frameworks for supporting to develop web application based on MVC architecture. These frameworks adopt name based rules for dispatching business logics depending on received requests. We can also implement

a web application easily by using Smarty (New Digital Group Inc., 2007), which is a template engine for PHP. However, it is impossible to separate View from Model or Logic completely because these are HTML based frameworks.

Struts (Apache Software Foundation, 2007c) is a famous framework for supporting Java based web application development. It also enables us to develop a MVC based web application easily by collaborating with JSTL (Apache Software Foundation, 2007b) and Velocity (Apache Software Foundation, 2007a). However, we have to manage a XML formalized configuration file called struts-config.xml to define dispatching rules, including requested URLs, Actions and JSPs. In addition, Struts does not support dependency injection. Therefore, unit testing and management of this configuration file are difficult in a large scale application development. Spring (Source, 2007) is a framework which includes DI container, libraries for presentation layer and database access layer.

However, we have to manage XML based configuration files like struts-config.xml in Struts framework. Seasar2 (The Seasar Foundation, 2007) is a DI Container which introduces convention over configuration concept. Therefore we do not have to manage configuration files even in a large scale application.

However, these frameworks and containers can not separate View completely by source code level.

On the other hand, Cairngorm (Adobe Labs, 2007) is a framework for supporting to develop Flex applications based on MVC architecture, however, it does not support dependency injection. Therefore developers have to define binding values or event handler directly in visible component tags, that is it does not support complete separation of View component.

8 CONCLUSIONS

We have presented a framework called Camel for development of RIAs. Camel has made it possible to separate View completely from any other components in source code level. That is, designers and developers do not have to share any source codes in application development processes, and it becomes easy not only to modify designs or logics separately but also to do testing an application.In addition, we have also introduced convention over configuration concept to reduce any configuration files to use this framework and shown the usage and utility of this framework by implementing a real application.

Finally, we would like to point out further issues to be resolved. In current implementation, Camel lacks validation mechanism for values bound to Model. We plan to introduce to extend validation mechanism of Flex with dependency injection. In addition, we usually make a mistake to type rule based names in development process. Therefore, we plan to provide a tool to complement these names as a Eclipse plug-in.

REFERENCES

37signals (2007). Ruby on Rail. http:// www.rubyonrails.org/.

Adobe Labs (2007). Cairngorm Framework. http://labs.adobe.com/wiki/index.php/Cairngorm.

Adobe Systems Inc. (2007a). Adobe Flash CS3 Professional. http:// www.adobe.com/products/flash/.

Adobe Systems Inc. (2007b). Adobe Flex2. http://www.adobe.com/products/flex/.

Allaire, J. (2002). Macromedia flash mx-a next-generation rich client. In *Technical report, Macromedia*.

Apache Software Foundation (2007a). Apache Velocity Project. http://velocity.apache.org/.

Apache Software Foundation (2007b). JSTL. http://jakarta.apache.org/taglibs/.

Apache Software Foundation (2007c). Struts. http://struts.apache.org/.

Driver, M., Valdes, R., and Phifer, G. (2005). Rich internet applications are the next evolution of the web. In *Technical report, Gartner*.

Duhl, J. (2003). White paper: Rich internet applications. In *Technical report, IDC*.

Ecma International (2005). Ecma script for xml specification.

Laszlo Systems Inc. (2007). Laszlo. http://www.laszlosystems.com.

Major Computing (2007). Mojavi Framework. http://www.mojavi.org/.

Microsoft Inc. (2007). Smart Client Developer Center. http://msdn.microsoft.com/smartclient/.

Mozilla.org (2007). XUL. http://www.mozilla.org/projects/xul/.

New Digital Group Inc. (2007). Smarty. http://smarty.php.net/.

Source, S. (2007). Spring Framework. http://www.springframework.org/.

Sun Microsystems Inc. (2007). Java Web Start Technology. http://java.sun.com/products/javawebstart/index.jsp.

The Seasar Foundation (2007). Seasar. http://www.seasar.org/.

Zend Technologies (2007). Zend Framework. http://framework.zend.com/.