ASPECTFX
A Framework for Supporting Collaborative Works in RIA by Aspect Oriented Approach

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

Keywords: Framework, Rich internet application, Software engineering, Aspect oriented programming.

Abstract: This paper presents AspectFX, a novel approach to enabling developers and designers to collaborate effectively in RIA development. Unlike traditional web applications, RIAs are implemented by a number of developers and designers; therefore it is reasonable to divide an application into modules and assign them to developers and designers, and collaborative works among them have been important. MVC architecture and OOP helps to divide an application into functional units as modules and bring efficiency to development processes. To play these modules as a single application, developers have to describe method invocations to utilize functionalities implemented in modules, however, developers need to describe additional method invocations that are not primary tasks for them. These additional method invocations make the dependencies among modules strong and these dependencies make it inefficient/difficult to implement and maintain an application.
This paper describes the design and implementation of AspectFX that introduces aspect-oriented concept and considers the additional method invocations as cross-cutting concerns. AspectFX provides methods to separate the cross-cutting concerns from primary concerns and weaves them for playing them as an application.

1 INTRODUCTION

Rich Internet Application (RIA) introduces the user experience of desktop applications. RIAs provide sophisticated user interfaces including attractive objects and animations for users to understand their status and results of the operations. In this way, compared to traditional web applications, it requires a number of developers and designers to implement an RIAs; therefore it is reasonable to divide an application into modules and assign them to developers and designers, and collaborative works among them have been important. However, dependencies among each module make it inefficient and difficult to implement and maintain an application even if we introduce MVC architecture and OOP. For example, although it is suitable for designers to create and manage animations including when animations should be started and stopped, developers usually manage them on behalf of designers because functionalities for these animations are provided as APIs.

Aspect-oriented programming (AOP)(Kiczales et al., 1997) is a new programming paradigm that separates cross-cutting concerns from primary concerns. The notion of AOP is suitable for the problems described above. Because the management of animations is not mainstream but cross-cutting concerns for for developers. Also, event handling is not mainstream but cross-cutting concerns for designers.

This paper describes a framework called AspectFX that makes it possible to rule out dependencies among modules and combine them to play a single application at runtime. AspectFX extends and leverages Flex framework(Adobe Systems Inc., 2009). In Flex, we can import and utilize animated objects created by Flash as components. AspectFX leverages this event model(Meier and Cahill, 2005) and dependency injection(Fowler, 2008) in order to weave cross-cutting concerns to mainstream at runtime. AspectFX also introduces name based conventions for detecting the points where cross-cutting concerns should be woven.

This paper is organized as follows. We first describe the background in section 2. We then explain the design and implementation in section 3 and behavior in section 4. In section 5, we conclude by providing summary and discussing future issues.

Fukuda H. and Yamamoto Y. (2010).
ASPECTFX - A Framework for Supporting Collaborative Works in RIA by Aspect Oriented Approach.
In *Proceedings of the 12th International Conference on Enterprise Information Systems - Databases and Information Systems Integration*, pages 398-402
DOI: 10.5220/0002906903980402

2 BACKGROUND

Almost all web applications are implemented based on MVC architecture nowadays. Therefore it is reasonable to implement RIAs based on the same architecture. In this section, we firstly explain MVC architecture in Flex. Then, we describe dependencies among modules in Flex applications.

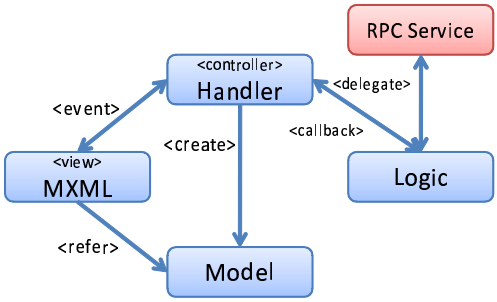

Figure 1: MVC architecture in Flex.

2.1 Flex Programming Model and MVC

Flex applications works by event-driven architecture. Therefore developers have to prepare event handlers and associate them with events dispatched from visible components. As shown in Figure 1, in general, it is reasonable to prepare a class called Handler that includes handler methods and correspond to an MXML that includes visible components. In the context of MVC architecture, as shown in Figure 1, MXML corresponds to *View* and Handler corresponds to *Controller*. In a handler method, developers implement business logics and create *Model* in order to reflect the result to MXML. In addition, it is recommended to prepare classes independent from this architecture and delegate business logics to them (we call these classes as Logic).

On the other hand, the triggers to start and stop animations embedded in a Flash component can be right before and after event handling. Consequently, we start and stop an embedded animation at the begginning and at the end of the handler method by invoking methods as Figure 2(a)-(i) and (iii). In addition, we also delegate business logics to another class by creating an instance and invoking a method in Figure2(a)-(ii).

2.2 Dependencies among Modules

It is also reasonable to assign developers and designers to each module for efficient RIA development; however dependencies among modules still make it inefficient and difficult. For example, designers mainly create MXML files for the design of an application and developers have to modify the MXML files to make associations between events and event handlers. Likewise, designers may modify event handlers implemented by developers to manage animations as shown in Figure 2(a).

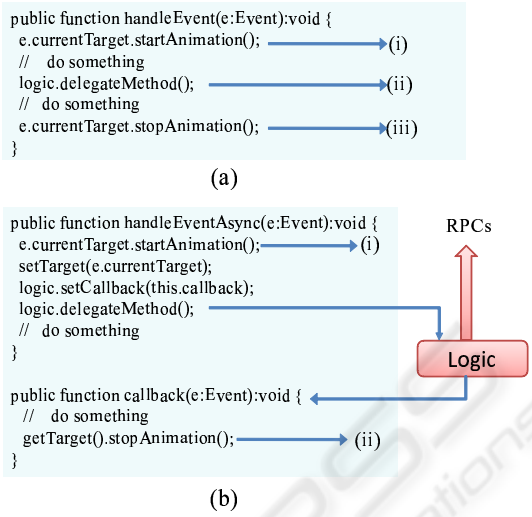

Figure 2: Pieces of code as cross-cutting concerns.

Besides, there will be more complicated dependencies between Handler and Logic. As we described in section 2.1, Logic is usually introduced for independent processes, especially RPCs, from specific architectures. As well as other components, RPC components adopt event-driven architecture. That is, when developers leverage RPC components, they have to prepare event handlers and add them to the RPC components. The handlers will be invoked by RPC components to notify the result of operations. If developers try to delegate a task to Logic that leverages RPCs, they have to prepare a callback method to create/modify Model that reflect the result of the task because the task is processed asynchronously. Moreover, if developers or designers try to apply an animation for the task, they have to specify method invocations for the animation in different methods as shown in Figure 2(b)-(i) and (ii).

3 SYSTEM IMPLEMENTATION

In AspectFX, we consider events in Flex as JoinPoint and leverage name based conventions as Pointcut. Besides, we also consider event handlers and methods to manage animations as Advice. AspectFX prepares two advice types such as “beforeAdvice” and “afterAdvice” to manage animations. These advice types need to be embedded in animated obejects as meth-

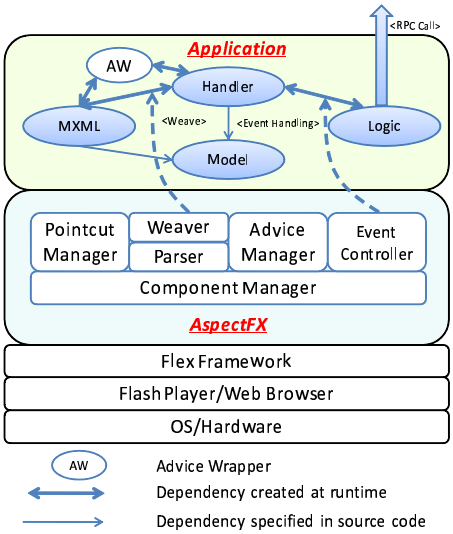

Figure 3: An architecture of AspectFX.

ods. In addition, AspectFX adopt event model to delegate tasks from Handler to Logic(we call this event as an application specific evnet). We show the architecture of AspectFX in Figure 3 and explain name based conventions.

3.1 Name based Conventions

AspectFX leverages convention over configuration concept to identify events and event handlers to be woven. Therefore AspectFX has several name based conventions for weaving. In addition, we assume Flash components are imported and utilized as visible components in MXML. Based on this assumption, we explain the conventions as follows.

1. Every component including Flash component defined as MXML tag must have “id” attribute and the value that can identify each component.
2. MXML and corresponded Handler have to be defined as MXML tag(Figure 4(a)). In addition, these modules must have the same parent in the composition of components(Figure 4(a)(b)).
3. MXML and corresponded Handler must have id attribute and the value of each. In addition, the prefix of each value must be the same to show they are relevant. Moreover the suffix of each must be “View” and “Handler” characters. For example, developers and designers should name each module with prefix “main” as “mainView” and “mainHandler”(Figure 4(a)).
4. In order to weave events and event handlers, the name of event handlers must be defined as “component id” + “event name” + “Handler” characters

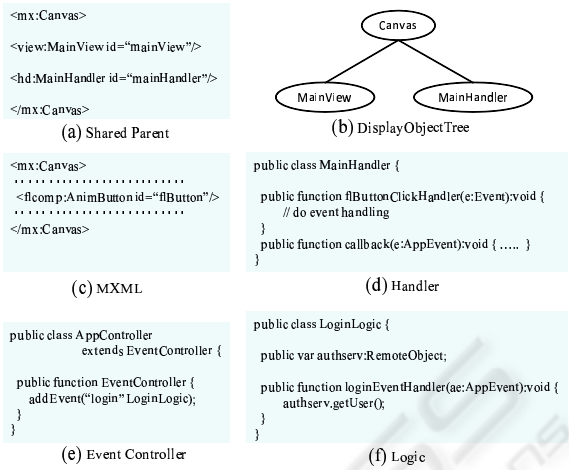

Figure 4: Name based conventions in AspectFX.

and “event name” must be capitalized. For example, to handler “click” event dispatched from a component named “flButton”, developers have to define “flButtonClickHandler” method in Handler(Figure 4(d)).

5. For leveraging animations, designer have to embed two types of advices in a Flash component as callback methods such as “beforeAdvice” and “afterAdvice”. “beforeAdvice” will be invoked before the event handling and “afterAdvice” is done after the event handling.
6. Developers need to prepare a class that extends Event Controller in order to leverage event model. They can use “addLogic” method to register a set of application specific event and Logic(Figure 4(e)). In addition, the name of event handlers defined in Logic that will handle application specific events must be defined as “event type” + “EventHandler” characters as shown in Figure 4(f)).

4 APPLICATION BEHAVIOR

In this section, we describe application’s behavior and also explain how to manage application specific events by introducing classes depicted in Figure 4.

4.1 Application Behavior

1. When an application starts, AspectFX confirms “id” attribute of each component and then if the suffix of the value is “View” characters, AspectFX get the reference of MainView and corresponded MainHandler by way of Canvas.

```
public class  AdviceWrapper {
  public var target:Object;
  public var callBack:Function;
  public var async:Boolean;

public function processAdvice(e:Event):void {
    target["beforeAdvice"].apply(null);
    callBack.apply(null,[e]);
    if(!async) target["afterAdvice"].apply(null,[e]);
  }
  public function processAdviceAsync(e:Event):void {
    if(async) target["afterAdvice"].apply(null);
  }
```

Figure 5: Implementation of Advice Wrapper.

2. Parser serializes MainView and MainHandler to XML expressions and parses them. Parser delegates Weaver to weave an event handler named "flButtonClickHandler" into "flButton".
3. If the visible component is a Flash component, Weaver creates an Advice Wrapper and sets the component (flButton) and corresponding event handler (flButtonClickHandler) into the Advice Wrapper. As shown in Figure 5, Advice Wrapper prepares event handler called "processAdvice" and Weaver injects the "processAdvice" into "flButton" to hook "click" event.
4. When a user clicks the flButton , the dispatched "click" event is caught by the processAdvice in Advice Wrapper. As shown in Figure 5, Advice Wrapper invokes "beforeAdvice" embedded the flButton first. Then the Advice Wrapper also invokes "flButtonClickHandler" that is set to a variable named "callBack". Finally, if "async" variable in Advice Wrapper is false, the Advice Wrapper invokes "afterAdvice". 4.2.

4.2 Application Specific Event Processing

To leverage application specific events, developers have to specify an extension class of Event Controller as an MXML tag as shown in Figure 6(a). Also, AspectFX provides "AppEvent" and "AppEventDispatcher" classes for application specific events.

We explain these processes as follows.

1. A "click" event is dispatched by user's operations. As described step 4 in section 4.1, Advice Wrapper invokes "beforeAdvice" and then invokes "flButtonClickHandler" defined in MainHandler. After that, if developers do not dispatch an event or do not specify a notified method in AppEvent, Advice Wrapper invokes "afterAdvice".

```
<ax:AsxApplication>
    <control:AppController id="appController"/>
</ax:AsxApplication>
```

(a)

```
public class MainHandler {

 public function flButtonClickHandler(e:Event):void {
   // do event handling
   var appEvent:AppEvent = new AppEvent("login", this,
                                "flButtonClickHandler", callback);
     AppEventDispatcher.dispatch(appEvent);
 }
 public function callback(e:AppEvent):void { ….. }
```

(b)

```
public class LoginLogic {

 public var authserv:RemoteObject;

 public function loginEventHandler(ae:AppEvent):void {
     // do business logic
     AppEventDispatcher.consume(ae);
 }
```

(c)

Figure 6: Source code for application specific event.

2. An application specific event is dispatched with an event type as "login" in flButtonClickHandler as shown in Figure 6(b).
3. Event Controller catches the "login" event and creates its corresponded LoginLogic. Then, Event Controller invokes "loginEventHanler" defined in LoginLogic.
4. An consume event is dispatched to notify the complete of the event. Event Controller invokes the notified method ("callback").
5. Event Controller sends a message to Advice Wrapper to notify the end of "login" event. Advice Wrapper invokes "afterAdvice" and then users can understand the process is completed.

5 CONCLUSIONS

In this paper, we have presented a framework called AspectFX for collaborative works in RIA development. We have also explained the design and implementation of AspectFX and confirm its availability via performance evaluations. AspectFX not only provides MVC based development process but also rules out pieces of code that make associations among modules from each source code by introducing AOP concept and event model. As a result, developers and designers are able to concentrate on their tasks. We believe these features of AspectFX help not only developers but also designers to develop and maintain an RIA application.

REFERENCES

Adobe Systems Inc. (2009). Adobe flex3. http://www.adobe.com/products/flex/.

Fowler, M. (2008). Inversion of Control Containers and the Dependency Injection pattern. http://www.martinfowler.com/articles/injection.html.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., and Irwin, J. (1997). Aspect oriented programming. In *In Proceedings of European Conference on Object-Oriented Programming (ECOOP)*.

Meier, R. and Cahill, V. (2005). Taxonomy of distributed event-based programming systems. *The Computer Journal*, 48(5):602–626.