УДК 004.4

Н.Н. Глибовец, И.В. Коваль, А.Н. Корень

СОЗДАНИЕ РАЗВИТЫХ ИНТЕРНЕТ-ПРОГРАММ НА ПРИМЕРЕ ПОСТРОЕНИЯ ВИДЖЕТ-ПРОГРАММ

Рассмотрено применение методологии создания развитых Интернет-программ на основе Adobe Flex 2.0 с применением паттернов проектирования и архитектурного каркаса. Подход проиллюстрирован построением виджет-программ.

Введение

Формирование нового подхода всегда ставит ряд вопросов, связанных с его эффективностью, целесообразностью, наличием преимуществ и недостатков, сферами применения. Не стала исключением технология развитых Интернет-программ (РИП) как новый шаг в эволюции Интернет-программ, отразивший постепенный, однако неизбежный переход веб-приложений от простой модели тонкого клиента к развитой модели распределенных функций [1].

Продемонстрируем применение технологии Adobe Flex 2.0. Разработчики Flex-приложений пишут код MXML и ActionScript в интегрированной среде разработки Flex Builder или в традиционных текстовых редакторах.

Flex framework – компонентно-ориентированный программный каркас разработки РИП для Flash Player среды исполнения [2], включающей MXML, ActionScript и библиотеку классов Flex. Предоставляет богатый набор расширяемых компонентов для интерфейса (более 100 компонентов и контейнеров), гибкую модель для контроля размещения и взаимодействия с пользователем, устойчивую к ошибкам инфраструктуру доступа к данным через удаленные сервисы, связывание этих данных с объектами данных и контроллерами интерфейса. Язык MXML (Multimedia eXtensible Markup Language) обеспечивает декларативный подход к созданию визуального представления РИП, а ActionScript 3.0 – мощный объектный язык программирования.

Библиотека классов Flex содержит набор классов, компоненты, менеджеры размещения и сервисные компоненты. Используя компонентно-ориентированную модель разработки Flex, разработчики могут использовать встроенные компоненты, создавать композитные компоненты из встроенных или создавать новые, расширяя встроенные компоненты или их классы.

Для достижения эффективности разработки Flex РИП, их масштабируемости и удобства поддержки, целесообразно учитывать паттерн разработки *модель-представление-контроль*. Удачным архитектурным каркасом для построения Flex РИП является Cairngorm, который базируется на MVC паттерне.

Использование предлагаемой методологии создания Flex РИП продемонстрируем на примере разработки РИП, которая предоставляет пользователю набор виджетов (часы, новости, калькулятор, блокнот, список задач и т.п.) и панель для управления ими. Как мини-программы, виджеты обеспечивают определенную функциональность. Далее для обозначения созданной РИП используем название "РИП персонализированных виджетов".

1. Создание и настройка проекта в Flex

Для создания любой Flex РИП целесообразно использовать интегрированную среду разработки Adobe Flex Builder 2, что позволяет работать с библиотеками классов Flex, языками программирования MXML и ActionScript 3.0 и компилятором для Flex-приложений. В Flex Builder все Flex-приложения содержатся в проектах. Для разработки РИП персонализирован-

ных виджетов использовался базовый тип проекта (Basic), предусматривающий доступ к данным с помощью XML или веб-сервисов на PHP / JSP / ASP.NET. Всего есть три типа проектов: Basic, ColdFusion Flash Remoting Service и Flex Data Services.

Визард для нового Flex-проекта автоматически генерирует конфигурационные файлы проекта, папку для результата проекта (bin), где будут размещаться скомпилированные SWF-файлы и основной файл проекта будет иметь расширение mxml.

Основной файл проекта сразу после создания будет иметь такое же название, как и проект, однако целесообразно переименовывать его в index.mxml.

Когда проект создан, необходимо подключить внешние библиотеки. РИП персонализированных виджетов базируется на архитектурном каркасе Cairngorm, поэтому надо подключить библиотеку cairngorm.swc. Библиотека cairngorm.swc общедоступна, ее можно взять из http://labs.adobe.com/wiki/index.php/Cairngorm:Downloads. Подключение библиотеки осуществляется в окне свойств проекта в разделе Flex Build Path на вкладке Library path с помощью кнопки Add SWC (Рис. 1 и 2).

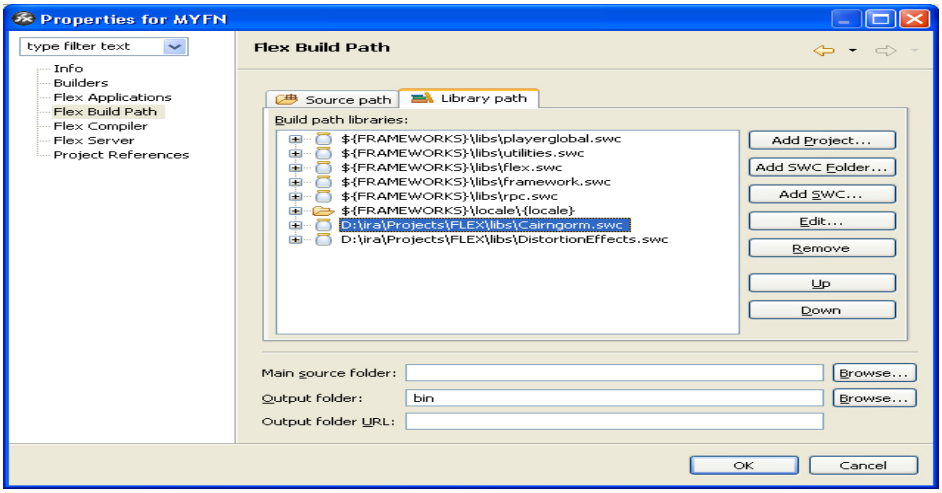

Рис. 1. Подключение библиотеки cairngorm.swc

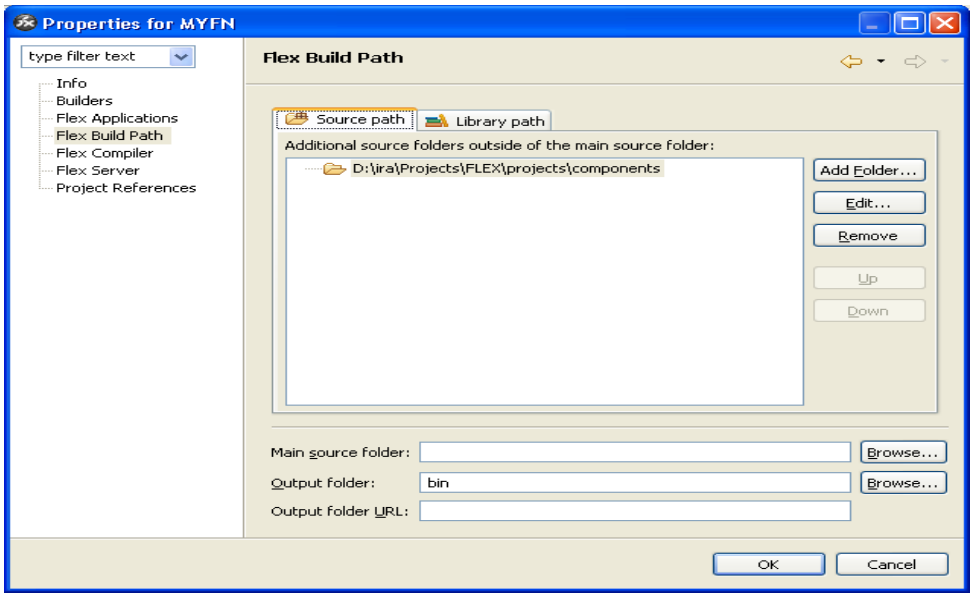

Рис. 2. Указание пути к дополнительным файлам с исходным кодом

Для реализации РИП персонализированных виджетов использовалась также общедоступная библиотека эффектов DistortionEffects.swc (http://weblogs.macromedia.com/auhlmann/archives/DistortionEffects.zip). Подключается аналогично библиотеке Cairngorm [3].

Проект Flex РИП может использовать дополнительные файлы с исходным кодом. Это один из способов обеспечения повторного использования компонентов. Для РИП персонализированных виджетов используются исходные файлы компонентов ResizableWindow и Debug (разработанные ранее), размещенные в отдельной папке под названием components. Пути к ним прописаны в окне свойств проекта в разделе Flex Build Path на вкладке Source path с помощью кнопки Add Folder.

Изначальное содержание index.mxml будет иметь следующий вид:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
</mx:Application>
```

2. Архитектура РИП персонализированных виджетов

В создании архитектуры РИП целесообразно руководствоваться рекомендациями архитектурного каркаса Cairngorm. Согласно документации Cairngorm API содержит ряд пакетов. Flex-проект имеет такую структуру папок:

- business – содержит классы, отвечающие за обмен данными с серверной частью, использование сервисов;
- commands – содержит классы для различных команд; классы команд реализуют интерфейс Command архитектурного каркаса и интерфейс IResponder (mx.rpc.IResponder);
- control – содержит классы событий, каждый из которых расширяет класс CairngormEvent архитектурного каркаса, и обработчик этих событий, расширяющий класс FrontController архитектурного каркаса Cairngorm;
- model – содержит классы, формирующие модель РИП, один из классов реализует интерфейс ModelLocator архитектурного каркаса;
- view – содержит набор классов, формирующих пользовательский интерфейс;
- vo – содержит классы с описаниями объектов для обмена между разными уровнями программы, (классы реализуют интерфейс ValueObject архитектурного каркаса).

Структуру проекта можно расширить другими папками. В частности проект РИП персонализированных виджетов содержит папку skin с файлом CSS для описания стилей Flex компонентов и с файлами SWF, содержащими графические элементы, из которых формируется визуальное представление каждого виджета и РИП в целом. Также есть папка constants с классом, который содержит все константы РИП.

3. Структура и взаимосвязи в РИП персонализированных виджетов

На структуру РИП персонализированных виджетов и взаимосвязи внутри РИП повлиял MVC паттерн и архитектурный каркас Cairngorm. Каждый из виджетов имеет свое представление, т. е. собственный MXML файл с описанием его вида и заданием внутренней функциональности, которое не влияет на состояние модели. Каждый виджет расширяет класс Widget, который в свою очередь реализует интерфейс IWidgetView.

С каждым виджетом связывается его контроллер, расширяющий класс WidgetController, что в свою очередь реализует интерфейс IWidgetController. Все контроллеры виджетов являются классами, входящими в состав модели РИП персонализированных виджетов. Контроллер содержит данные, необходимые виджету, и отвечает за связь виджета с остальными элементами РИП.

Для нового виджета используем WidgetFactory, реализующий паттерн Factory.

Виджет может инициировать ряд событий. Каждому событию соответствует свой класс. Например, OpenWidgetEvent, RemoveWidgetEvent, SaveWidgetDataEvent

используются всеми виджетами РИП. Однако специальные события можно инициировать только определенным виджетом, например, LoadRadioDataEvent, LoadNewsEvent, LoadCalendarDataEvent. Все события обрабатывает класс MYFNFrontController.

В результате конкретного события выполняется заранее определенная команда. Каждая команда связана с обращением к серверу и имеет три *public* метода: *execute, result* и *fault*. Метод *execute* содержит инструкции, которые необходимо выполнить о передаче полномочий делегату веб-сервиса. Метод *result* вызывается, когда успешно получен результат взаимодействия РИП с сервисом, а метод *fault* – в случае неудачи (содержит код ошибки).

Все доступные для РИП сервисы описаны в Services.mxml, расширяющем класс ServiceLocator архитектурного каркаса Cairngorm. Делегаты (Delegate) отвечают за обращение к конкретному сервису, передачу ему параметров и связывание результата взаимодействия сервиса с командой, которая делегировала вызов сервиса.

Активизация виджета в РИП и его удаление осуществляется посредством панели управления виджетами. Некоторые из виджетов, такие как радио и калькулятор, могут иметь только один экземпляр, остальные - произвольное количество. Модель РИП персонализированных виджетов реализует механизм отслеживания активных виджетов.

4. Схема добавления виджета

Рассмотрим схему добавления нового виджета *радио* для иллюстрации аспектов, описанных в предыдущем разделе. Прежде всего в папке view для нового виджета создаем представление как описание графического отображения RadioWidget.mxml посредством языка MXML. Функции RadioWidget.mxml создаются посредством языка ActionScript 3.0 и размещаются в тезисе Script:

```
<mx:Script>
      <![CDATA[
      // код мовою ActionScript
      ]]>
      </mx:Script>
```

Для представления виджета создается контроллер RadioController.as в папке model/widgets. К константам РИП в MYFNConstansts.as добавляется новая константа, соответствующая названию нового виджета для его идентификации:

```
public static const WID-
GET_RADIO:String = "RadioWidget";
```

В WidgetFactory.as добавляется возможность создания нового виджета радио.

К DashboardView.mxml, содержащему панель управления виджетами, добавляется новая кнопка, отвечающая за активацию виджета радио. С каждой кнопкой, которая является экземпляром класса ToolbarIcon.mxml, связано событие OpenWidgetEvent, параметром которого является название виджета.

```
private function clickI-
con(name:String):void {
          var
event:OpenWidgetEvent = new
OpenWidgetEvent(name);

CairngormEventDispatcher.getInsta
nce().dispatchEvent(event);}
```

Конструктор классса OpenWidgetEvent, расширяющий класс CairngormEvent, имеет следующий вид:

```
public function
OpenWidgetEvent
(widgetName:String, wid:String =
null) {
          super(MYFNFrontControll
er.OPEN_WIDGET);
          this.widgetName =
widgetName;
          this.wid = wid;}
```

Класс MYFNFrontController обрабатывает событие OpenWidgetEvent, с которым связана команда OpenWidgetCommand:

```
public function
MYFNFrontController() {
          addCommand(OPEN_WIDGET,
OpenWidgetCommand);
          ...}
```

Для радио необходимо загрузить с сервера список радиостанций, которые будут проигрываться с помощью виджета. С этой целью следует создать событие LoadRadioDataEvent и соответствующую команду LoadRadioDataCommand.

Класс LoadRadioDataEvent имеет вид:

```
 package control
 {   import
model.widgets.IWidgetController;
     import
com.adobe.cairngorm.control.Cairn
gormEvent;
          public            class
LoadRadioDataEvent        extends
CairngormEvent
     {         public        var
widgetController:IWidgetControlle
r;
     public              function
LoadRadioDataEvent
(widgetController:
     IWidgetController) {
     su-
per(MYFNFrontController.LOAD_RADI
O_DATA);
     this.widgetController      =
widgetController;
          }    }    }
```

Класс LoadRadioDataCommand может описать следующим образом:

```
     package commands
     {    import
com.adobe.cairngorm.commands.Comm
and;
          import
mx.rpc.IResponder;
          ...
          import
model.MYFNModelLocator;
               public      class
LoadRadioDataCommand   implements
Command, IResponder
          {         private  var
widgetController:RadioController;
               public   function
execute(event:CairngormEvent):voi
d         {
          widgetController      =
RadioController(
LoadRadioDataEvent(event).widgetC
ontroller);
                    var
delegate:LoadRadioDataDelegate   =
new
LoadRadioDataDelegate(this);
                    var
authTicket:String                 =
MYFNModelLocator.
     getInstance().authTicket;
          delegate.loadService(au
thTicket); }
               public    function
result(data:Object):void
               {    try {
                         var
xml:XML          =          MYFNU-
tils.strToXML(data.result);
     widgetController.resultStati
ons(
     xml.LoadRssResult);
     } catch (e:Error) {
          Debug.write(e.message);
               }         }
               public    function
fault(info:Object):void    {
          De-
bug.write(FaultEvent(info).fault.
faultString);
               }    }    }
```

Инициализация события LoadRadioDataEvent происходит в методе loadStations класса RadioController с помощью CairngormEventDispatcher:

```
     public function loadStations
():void {
          var
event:LoadRadioDataEvent   =   new
LoadRadioDataEvent(this);
     CairngormEventDispatcher.get
Instance().dispatchEvent(event);}
```

После инициализации событие будет обработано в MYFNFrontController. Он определяет, какую команду выполнить при условии наступления того или иного события. Поэтому в MYFNFrontController необходимо добавить следующий код:

```
     public      static      var
LOAD_RADIO_DATA:String   =   "load
radio data";
     public              function
MYFNFrontController() {
               ...
          addCommand(LOAD_RADIO_D
ATA, LoadRadioDataCommand);
               ...}
```

Таким образом создано событие и добавлен код, который отвечает за его инициализацию. Обработчик событий будет анализировать новое событие, связывая с ним созданную команду. Теперь переходим к созданию делегатора LoadRadioDataDelegate, вызываемого командой LoadRadioDataCommand. Сначала следует задать сервис, который будет использовать делигатор. Описать сервис не-

обходимо в Services.mxml, используя такой код, к примеру:

```
      <mx:WebService
id="rssService"
      wsdl="http://www.flexmonster
.com/myfn/RssService.asmx?wsdl"
      showBusyCursor="true">
            <mx:operation
name="LoadRss"
resultFormat="xml">
                  <mx:request>

      <authTicket></authTicket>
                        <url></url>

                        <login></login>

                        <password></password>
                              </mx:request>
                        </mx:operation>
                  </mx:WebService>
```

Делегат LoadRadioDataDelegate отвечает за обращение к сервису rssService. Он обеспечивает передачу параметров authTicket, url, login и password сервису и связывание результата взаимодействия сервиса с командой, которая делегировала вызов сервиса. Ниже приведен код LoadRadioDataDelegate:

```
package business
{
import mx.rpc.IResponder;
import com.adobe.cairngorm.business.ServiceLocator;
import mx.rpc.AsyncToken;
import constants.MYFNConstansts;
import mx.rpc.soap.WebService;

public class LoadRadioDataDelegate
{
      protected var command:IResponder;
      protected var service:Object;

      public function LoadRadioDataDelegate
                  (command:IResponder) {
            this.service = ServiceLocator.getInstance().
                  getWebService("rssService") as WebService;
            this.command = command;
      }

      public function loadService(authTicket:String):void {
            var url:String = MYFNConstansts.RADIO_LINK;
            var login:String = MYFNConstansts.RADIO_LOGIN;
            var password:String = MYFNConstansts.RADIO_PASSWORD;
            service.loadWSDL();
      var token:AsyncToken = service.LoadRss(authTicket, url, login,
password);
            token.addResponder(command);
            }
      }
}
```

Так происходит добавление нового виджета в разработанную РИП.

Выводы

Развитые Интернет-программы как новый подход к созданию веб-систем, которые реализуют модель клиент-серверной архитектуры с толстым клиентом. Они используют преимущества асинхронного режима взаимодействия с сервером, развивают сервисно-ориенти-рованную архитектуру и самого сервисно-ориентированного клиента.

Комплексным решением для построения развитых интернет-программ видится технология Adobe Flex 2.0, на базе которой создана методология разработки РИП, продемонстрированная на примере

разработки развитой интернет-программы, предоставляющей пользователю набор виджетов (часы, новости, калькулятор, блокнот, список задач) и панель для управления ими.

1. *Driver*. M., Valdes. R., Phifer. G. Rich Internet Applications Are the Next Evolution of the Web // Technical report, Gartner. – 2005. – P. 6–23.
2. *Webster*. S. Developing Flex RIAs with Cairngorm Microarchitecture / S. Webster // Adobe Consulting [Electronic resource]. – Available at: http://www.adobe.com/devnet/flex/articles/cairngorm_pt1.html
3. *Flex* Application Design for Cairngorm [Electronic resource]. – Available at: http://www.digimmersion.com/support/Flex%20UI%20Design %20For%20Cairngorm.pdf
4. *Colin* M. Essential ActionScript 2.0 // O'Reilly. – 2004. – 544 p.

Получено 06.04.2011

Про авторів:

Глибовець Микола Миколайович,
доктор фізико-математичних наук,
професор, декан факультету інформатики Національного університету «Києво-Могилянська Академія»,

Корень Олександр Миколайович,
пошукувач,

Коваль Ірина Володимирівна,
магістр.

Місце роботи авторів:

Національний університет
«Києво-Могилянська Академія»,
04071, Київ, вул. Г. Сковороди, 2.
Тел.: 067 209 0740.
glib@ukma.kiev.ua