

RIA -> "Flex in Action"

GRUPO 3

INTEGRANTES:

- Ana Lia Castellanos
- Yesica Cortes
- Pablo de Sosa
- Federico Garcia
- Carmen Barreira

INTRODUCCIÓN A FLEX

- Esencialmente Flex es un conjunto de librerías – o framework – para desarrollo de UI (user interface).
- Es el entorno de trabajo para crear aplicaciones en lenguaje Action Script
- Está específicamente creado para RIA y es compatible con DB en XML y más

Capítulo 14 → Implementar vista estados

Con vista estados, es posible construir RIAs que imparten una experiencia de usuario pulida y satisfactoria.

14.1 Comprender estados

Las vistas pueden tomar diferentes apariencias visuales y representaciones en función de las condiciones en que están presentadas.

El estado de vista se puede definir como una apariencia visual particular, con un comportamiento, y la representación para una vista (UI).

Las vistas tienen al menos un estado, que es conocido como el **estado por defecto o la base estado de la vista**. Se puede definir cualquier número de estados adicionales en función de las necesidades.

Las vistas de estados pueden ser utilizados para muchos otros escenarios, tales como los siguientes:

- Una vista de búsqueda que muestra los resultados en un estado diferente al formulario de búsqueda
- Una vista de diseño-personalización que permite a los usuarios personalizar su aplicación de aspecto definiendo diferentes diseños como diferentes estados
- Vista que muestra datos de usuario en un estado y luego permite la edición en diferentes estados.

14.2 Estados de Flex

En Flex, cada aplicación, componente personalizado, o la vista tiene al menos un estado, que es conocido como el estado de la vista por defecto. Se puede definir estados adicionales agregando un estado a la Propiedad estados.

El estado por defecto es el estado original de los componentes de código en la aplicación.

El siguiente código muestra cómo se pueden definir dos estados adicionales:

```
<s:states>
<s:State name="somestate" />
<s:State name="state2" />
<s:State name="state3" />
</s:states>
```

El código anterior crea tres estados: **somestate**, **estado2** y **Estado3**. Flex 4 define el estado por defecto para el primer estado en la lista. En este caso, somestate es el estado por defecto.

También puede controlar el estado predeterminado utilizando la propiedad currentState de cualquier vista (la vista es la aplicación o componente personalizado). En el ejemplo anterior, sólo hemos creado un estado por su nombre, que en última instancia, no significa nada hasta que se hace referencia a él.

La tabla 14.1 muestra las propiedades relacionadas con el estado demx.core.UIComponent.

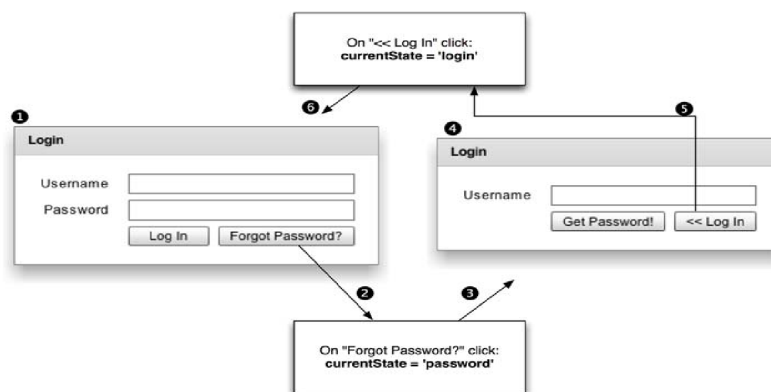
Table 14.1 View state-related properties of the `mx.core.UIComponent` class

Property	Type	Description
<code>states</code>	Array	An array of state objects. This is populated to create more view states or read from to find available states.
<code>currentState</code>	String	The current view state name. The default value is '' or null (default view state).
<code>currentCSSState</code>	String	The current state used for CSS pseudo selectors. The default value is the same as the <code>currentState</code> property.

Table 14.2 View state-related events of the `mx.core.UIComponent` class

Property	Type	Description
<code>currentStateChanging</code>	<code>StateChangeEvent</code>	This event is dispatched after the <code>currentState</code> property has been changed but before the view state changes. This event can be used to validate the change request or to perform actions before the state changes. It can also be used to prevent the state from changing.
<code>currentStateChange</code>	<code>StateChangeEvent</code>	This event is dispatched when the view's state has been changed.

Ejemplo:



La vista se compone de dos diferentes estados: inicio de sesión y contraseña. Una vista consiste en los estados, ya sea uno o múltiples estados.

La figura 14.3 muestra el flujo entre los dos estados. No olvidar que esto es el mismo componente pero con dos estados diferentes ejecutados haciendo clic en un botón.

14.2.1 Trabajar con propiedades

El listado 14,1 es un ejemplo sencillo que muestra dos botones configurados para conmutar el estado de la aplicación.

Listado 14.1 Changing properties with states

Listing 14.1 Changing properties with states

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/halo">

  <s:layout>
    <s:VerticalLayout />
  </s:layout>
  <s:states>
    <s:State name="orange" />
    <s:State name="black" />
  </s:states>

  <s:HGroup>
    <s:Button label="Orange" click="currentState = 'orange'" />
    <s:Button label="Black" click="currentState = 'black'" />
  </s:HGroup>
  <s:Rect width="200" height="200">
    <s:fill>
      <s:SolidColor color.black="#000000" color.orange="#de7800" />
    </s:fill>
  </s:Rect>
</s:Application>
```

State declaration named orange

State declaration named black

Orange button changes state to orange

Black button changes state to black

200x200 rectangle

Rectangle's fill, color changed by state

En el listado 14.1, hay dos estados: naranja y negro. Los dos botones tienen un evento clic fijado para cambiar la propiedad `currentState` de la solicitud. Cuando en cualquiera de los botones se hace clic, para cambiar el estado de la aplicación, los cambios de color del `Rect` se hacen con la siguiente línea de código:

```
<s:SolidColor color.black="#000000" color.orange="#de7800" />
```

La clase `SolidColor` tiene una propiedad de color. Esta propiedad puede ser configurada normalmente, pero en otro caso se selecciona propiedad basándose en que el estado, con la siguiente sintaxis: `property.state = "value"`.

La figura 14.4 muestra los cambios de estado a estado. El lado izquierdo de la figura muestra el estado original, naranja, y el lado derecho muestra el segundo estado, negro.

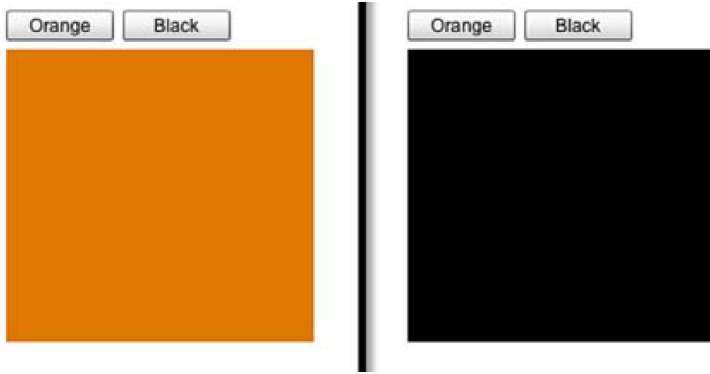


Figure 14.4 Two different states from listing 14.1

La siguiente línea de código establece como cambiar la propiedad label de cada estado:

```
<s:Button label.orange="Black" label.black="Orange">
```

Cuando currentState la solicitud se ajusta a la naranja, la etiqueta se leerá "Negro" y viceversa para el estado negro.

14.2.2 Utilización de controladores de eventos

El listado 14,3 muestra cómo utilizar los eventos específicos de cada estado.

Listing 14.3 Using state-specific events

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/halo">

  <s:layout>
    <s:VerticalLayout />
  </s:layout>
  <s:states>
    <s:State name="orange" />
    <s:State name="black" />
  </s:states>
  <s:Button label.orange="Black" label.black="Orange"
            click.orange="currentState='black'"
            click.black="currentState='orange'"/>
  <s:Rect width.orange="200" height.orange="200"
          width.black="150" height.black="150">
    <s:fill>

      <s:SolidColor color.black="#000000" color.orange="#de7800" />
    </s:fill>
  </s:Rect>
</s:Application>
```

State declarations

Different label per state

State click handlers

Rectangle dimensions per state

Rectangle's solid fill, color changed by state

Esta lista proporciona pocas novedades de las listas anteriores. Hay un botón de alternar el estado de aplicación de naranja a negro y viceversa. En cada estado de llenado del Rect cambia del color # de7800 a # 000000. La gran diferencia aquí es al hacer clic en el botón del controlador.

El siguiente código, es el código del botón.

```
<s:Button label.orange="Black" label.black="Orange" click.orange="currentState='black'"  
click.black="currentState='orange'"/>
```

La etiqueta del botón cambia de la misma forma que antes, pero el controlador también utiliza la misma sintaxis que las propiedades específicas del estado: **event.state = "valor"**. Las mismas reglas aplicables a las propiedades específicas del estado se aplican aquí también.

Además, también se puede escribir el código del botón de la siguiente manera:

```
<s:Button label.orange="Black" label.black="Orange"  
click="currentState='black'"  
click.black="currentState='orange'"/>
```

El naranja es el caso por defecto, por lo que no necesariamente se tiene que establecer explícitamente el controlador de eventos naranja. El código del botón también se podría utilizar como controladores de eventos en un script. El valor de controlador de eventos utiliza actualmente en línea ActionScript, lo cual es correcto.

14.2.3 Utilización de grupos estatales

Grupos estatales proporcionan una forma limpia para la creación de un grupo de Estados, lo que permite hacer referencia a ellos como los estados de las propiedades específicas del estado, eventos y similares. La sintaxis es la misma que para las propiedades o eventos:

<property de event> <state o <grupo = "valor".

Para crear un grupo de estados, se hace referencia al grupo en la declaración específica del estado, así:

```
<s:states>  
<s:State name="orange" stateGroups="box" />  
<s:State name="black" stateGroups="box" />  
<s:State name="green" stateGroups="circle" />  
<s:State name="blue" stateGroups="circle" />  
</s:states>
```

Se han creado dos grupos de estados por separado: caja y círculo. La propiedad **stateGroups** del componente estatal es una lista separada por comas. Esto significa que puede tener un estado que puede ser parte de múltiples grupos.

Por ejemplo, el siguiente código muestra cómo tener un Estado principal en múltiples grupos:

```
<s:State name="orange" stateGroups="box,circle" />
```

Diferencia entre un grupo y un estado.

Un estado es un estado real. Esto significa que se puede establecer la propiedad currentState a un Estado con el fin de ajustar su punto de vista de alguna manera.

Un grupo no es un estado sino más bien una forma de disminuir la cantidad de MXML necesario para cambiar estados.

El listado 14.5 muestra un ejemplo del uso de grupos estatales.

Listing 14.5 Using state groups

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/halo">

  <s:layout>
    <s:VerticalLayout />
  </s:layout>
  <s:states>
    <s:State name="orange" stateGroups="box" />
    <s:State name="black" stateGroups="box" />
    <s:State name="green" stateGroups="circle" />
    <s:State name="blue" stateGroups="circle" />
  </s:states>
  <s:Button label.orange="Black" label.black="Green"
            label.green="Blue" label.blue="Orange"
            click.orange="currentState='black'"
            click.black="currentState='green'"
            click.green="currentState='blue'"
            click.blue="currentState='orange'" />
  <s:Rect width="200" height="200"
          visible="false" includeInLayout="false"
          visible.box="true" includeInLayout.box="true">
    <s:fill>

      <s:SolidColor color.black="#000000" color.orange="#de7800" />
    </s:fill>
  </s:Rect>
  <s:Ellipse width="200" height="200"
             visible="false" includeInLayout="false"
             visible.circle="true" includeInLayout.circle="true">
    <s:fill>
      <s:SolidColor color.green="green" color.blue="blue" />
    </s:fill>
  </s:Ellipse>
</s:Application>
```

States in box group

States in circle group

Rectangle to show in box group

Ellipse to show in circle group

El Rect tiene los mismos colores de relleno, pero es necesario cambiar la visibilidad, ya que sólo desea mostrar el Rect cuando la currentState es un estado del grupo caja. Aquí es donde se utiliza el cuadro de grupo.

El Rect tiene dos propiedades específicas del grupo de casillas: visible and includeInLayout.

La sintaxis para cambiar la visibilidad de la propiedad currentState del Estado caja:

visible. box = "true".

Es exactamente como un estado, pero nunca se ve currentState = "caja" en el código.

El grupo estado actual está determinado por lo que el estado está seleccionado y que grupos pertenece el estado seleccionado.

La visibilidad Rect diría por ejemplo: Para todos los estados no incluidos en el grupo de estado de caja,

ocultar este componente y para todos los estados incluidos en el grupo de estados cuadro, muestran este componente.

La figura 14.7 muestra el resultado de la inclusión de 14.5. La misma muestra todos los estados separados por líneas. Los dos primeros estados pertenecen al grupo de caja y este último al grupo círculo. Tanto la elipse y el Rect que hagan lo exactamente lo mismo, se esconden para un grupo determinado estado, y Flex determina el grupo apropiado en función de la lista stateGroups del currentState.

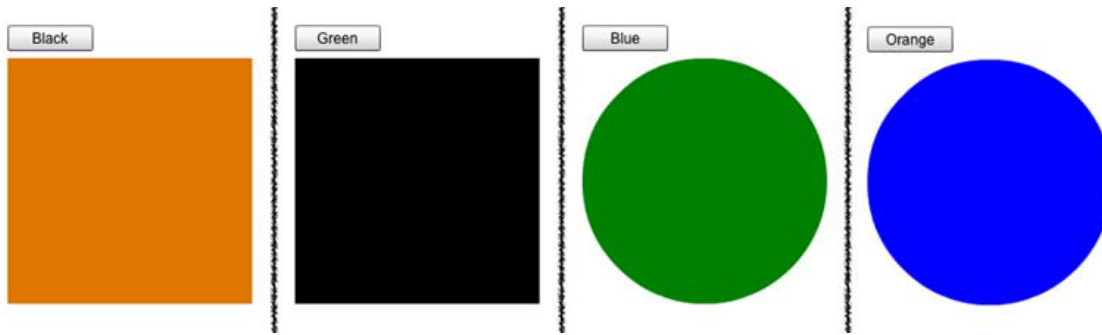


Figure 14.7 Simultaneous look at views in listing 14.5

14.2.4 Adición y eliminación de componentes

El gran truco para que la adición y eliminación no sea difícil está en saber si el elemento está disponible cuando se lo necesita en el código, es decir, si no hay un elemento disponible y trato de acceder a él, obtendré un error de ejecución. También hay que comprobar si realmente quiero el elemento eliminado o si deseo que este escondido en la pantalla. Una regla general es crear componentes del sistema sólo cuando se los necesita y los destruirlos cuando no se necesitan más.

Para añadir o eliminar un elemento, se utilizan las propiedades **excludeFrom** **includeIn** y los datos están en listas delimitadas por comas de los estados, pero no se pueden utilizar juntos.

El siguiente código muestra cómo se puede incluir y excluir elementos en estados específicos:

```
<s:Button includeIn="myState" />  
<s:Button includeIn="yourState" />
```

El listado 14.6 muestra cómo se pueden incluir en lugar de alternar la visibilidad.

Listing 14.6 Using includeIn

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/halo">

  <s:layout>
    <s:VerticalLayout />
  </s:layout>
  <s:states>
    <s:State name="orange" stateGroups="box" />
    <s:State name="black" stateGroups="box" />
    <s:State name="green" stateGroups="circle" />
    <s:State name="blue" stateGroups="circle" />
  </s:states>
  <s:Button label.orange="Black" label.black="Green"
            label.green="Blue" label.blue="Orange"
            click.orange="currentState='black'"
            click.black="currentState='green'"
            click.green="currentState='blue'"
            click.blue="currentState='orange'" />
  <s:Rect width="200" height="200" includeIn="box">
    <s:fill>
      <s:SolidColor color.black="black" color.orange="#de7800" />
    </s:fill>
  </s:Rect>
  <s:Ellipse width="200" height="200" includeIn="circle">
    <s:fill>
      <s:SolidColor color.green="green" color.blue="blue" />
    </s:fill>
  </s:Ellipse>
</s:Application>
```

Include Rect in box group

Include Ellipse in circle group

Eliminando cuatro líneas enteras de código mediante la sustitución de las líneas con un simple includeIn lo lee: Incluir el Rect en toda la caja grupos estatales y excluirlo de cualquier otro Estado no en el grupo de caja. También se puede utilizar excludeFrom para el Rect y Ellipse mediante el intercambio los valores, así:

```
<s:Rect width="200" height="200" excludeFrom="circle">
<s:fill>
<s:SolidColor color.black="black" color.orange="#de7800" />
</s:fill>
</s:Rect>
<s:Ellipse width="200" height="200" excludeFrom="box">
<s:fill>
<s:SolidColor color.green="green" color.blue="blue" />
</s:fill>
</s:Ellipse>
```

IncludeIn y excludeFrom tanto destruyen y crean el elemento de destino en consecuencia.

Las políticas de creación y destrucción permiten determinar si el componente es creado y / o destruido.

Las siguientes dos líneas son las huellas de encargo de Including2, el nombre de nuestro de la aplicación, que muestra el evento complete, aplicación y seguimiento del boxElement y circleElement. Observe que el boxElement tiene un componente de los padres y circleElement no lo hace, boxElement tiene un padre porque se ha añadido a la etapa, pero círculo no es así, pero existe a causa de la política de creación de configurar: artículo-CreationPolicy = "inmediata". Esto le dice a la aplicación para crear este elemento secundario de inmediato, permitiendo el acceso al elemento en el inicio. Si la creación política se establece en diferido, se creó el elemento sólo cuando se necesita. Como se muda al estado siguiente, la traza muestra lo siguiente:

```
_Including2_Button1 click  
boxElement removedFromStage  
boxElement addedToStage
```

El primer trazo es el clic en el botón. A continuación, el boxElement se elimina de la fase y después añadirlo al escenario.

El siguiente estado es una parte del grupo de círculo, así que veremos unas líneas más:

```
_Including2_Button1 click  
boxElement removedFromStage  
circleElement addedToStage  
circleElement creationComplete
```

El click del botón cambia el estado a verde, y se puede ver que el boxElement es retirado y luego se añade el círculo y la creación complete se distribuye.

El circleElement es ahora visible y el boxElement se ha ido. Si rastreamos boxElement, mostraría nulo, ya que ha sido completamente eliminado. Esto es importante debido a que su código ya no puede hacer referencia a boxElement porque se ha destruido.

```
_Including2_Button1 click  
circleElement removedFromStage  
circleElement addedToStage
```

Se retira el circleElement y después se añadió.

Valores disponibles para cada política

La Tabla 14.3 muestra los valores disponibles para cada política.

Table 14.3 Available policies

Policy	Values	Description
itemCreationPolicy	deferred	Create the element only when it's needed.
	immediate	Create the element immediately.
itemDestructionPolicy	auto	Let Flex determine when to destroy.
	never	Never destroy the element.

14.2.5 Componentes reparentalización

Reparent es un elemento que puede utilizar ActionScript: ***newParent.addChild (myElement)***.

Listado 14.8 muestra Reparent en acción.

Listing 14.8 Reparenting elements

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/halo">

  <s:layout>
    <s:VerticalLayout paddingLeft="20" paddingTop="20" gap="20" />
  </s:layout>
  <s:states>
    <s:State name="boxLeft" />
    <s:State name="boxRight" />
  </s:states>
  <s:HGroup>
    <s:HGroup gap="20">
      <s:VGroup id="left" width="200" height="200">
        <s:Label text="Left" />
        <s:Button id="button" includeIn="boxLeft"
                  label="box >>"
                  label.boxRight="<< box"
                  click.boxLeft="currentState='boxRight'"
                  click.boxRight="currentState='boxLeft'" />
        <s:Rect id="boxElement" width="200" height="200"
                includeIn="boxLeft">
          <s:fill>
            <s:SolidColor color="#de7800" />
          </s:fill>
        </s:Rect>
      </s:VGroup>
      <s:VGroup id="right" width="200" height="200">
        <s:Label text="Right" />
        <fx:Reparent target="boxElement" includeIn="boxRight" />
        <fx:Reparent target="button" includeIn="boxRight" />
      </s:VGroup>
    </s:HGroup>
  </s:HGroup>
</s:Application>
```

State declarations

Parent 1 (left side)

Button to toggle state

Rect to move between sides

Parent 2 (right side)

boxElement placeholder

Button placeholder

Como muestra la figura 14.8, la vista inicial consiste en una etiqueta, el botón de activación y el rectángulo naranja. Lo que hay que tener en cuenta es el uso de `includeIn` del rectángulo.

Se dice que Flex desea incluir el rectángulo en el estado `boxLeft`, es decir, que está excluido de todos los demás estados, incluyendo `boxRight`. Cuando hacemos clic en el botón, el rectángulo se elimina de la etapa y, dependiendo de la política de destrucción, eliminado, sino porque tiene un componente `Reparent` dirigidas a la caja, el elemento se elimina de `boxLeft` y se añade a `boxRight`. Lo mismo ocurre con el botón, ya que está incluido en `boxLeft` y tiene un `Reparent` incluido, `boxRight`.

`<fx:Reparent target="boxElement" includeIn="boxRight" />`

```
<fx:Reparent target="button" includeIn="boxRight" />
```

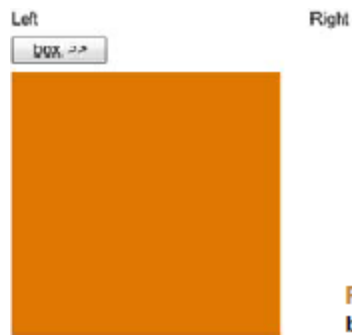


Figure 14.8 Initial view with the button and box on the left

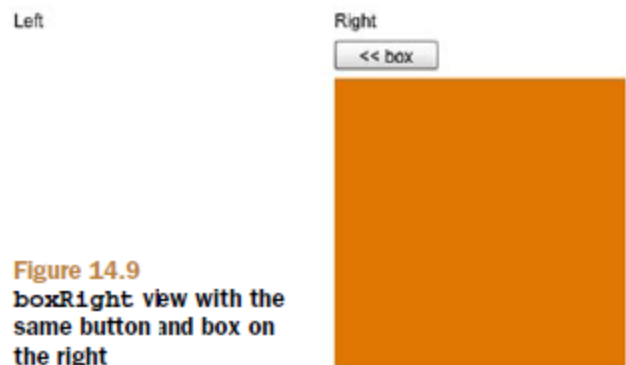


Figure 14.9 `boxRight` view with the same button and box on the right

Lo que está sucediendo aquí es específico del Estado. Cuando el `CurrentState` hace cambios `boxRight`, mueva el `boxElement` y el botón de este padre.

NOTA: Siempre hay que tener en cuenta el caso de uso de `Reparent`. La idea es cambiar el padre de un componente cuando se entra en un estado diferente. Esto significa que no puede tener el componente en dos lugares a la vez, por lo que todas las instancias `Reparent` deben hacer referencia a un estado diferente de la que el componente ya reside.

14.2.6 Eventos estatales

Listing 14.9 Listing for state events

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/halo"
               currentStateChange="log('change: '+currentState);"
               currentStateChanging="log('change: '+currentState);">
  <s:layout>
    <s:VerticalLayout />
  </s:layout>
  <s:states>
    <s:State name="orange"
            enterState="log('--orange enter')"
            exitState="log('--orange exit')" />
    <s:State name="black"
            enterState="log('--black enter')"
            exitState="log('--black exit')" />
  </s:states>
  <fx:script>
    <![CDATA[
      protected function log(text:String):void{
        logElement.text += text + "\n";
      }
    ]]>
  </fx:Script>
  <s:HGroup>
    <s:Button label.orange="Black" label.black="Orange"
            click.orange="currentState = 'black'"
            click.black="currentState = 'orange'" />
  </s:HGroup>
  <s:Rect width="300" height="230">
    <s:fill>
      <s:SolidColor color.black="#000000" color.orange="#de7800" />
    </s:fill>
  </s:Rect>
  <s:TextArea id="logElement" width="300" height="150" />
</s:Application>
```

Handle state change (points to `currentStateChange`)

Handle the state changing (points to `currentStateChanging`)

Handle orange's enter and exit states (points to `enterState` and `exitState` of the `orange` state)

Handle black's enter and exit states (points to `enterState` and `exitState` of the `black` state)

En la aplicación tenemos dos eventos: `currentStateChange` y `currentStateChanging`. Tabla 14.2 proporciona descripciones detalladas de los dos eventos, pero le ocurre durante los dos eventos es que se conecte un texto más la propiedad `currentState` por lo que se puede ver cuando se producen y cuál es el estado en el momento de cada evento.

Cada declaración de estado también detecta el `enterState` y eventos `exitState`. Esto indica que cuando entre o salga de un estado específico, respectivamente. Las imágenes en la aplicación son las mismas que antes, pero con un rectángulo ligeramente más ancho y un `TextArea` para el registro. La figura 14.10 muestra la aplicación después de cambiar de color naranja (estado predeterminado) a negro y de vuelta a la naranja.



Figure 14.10 Application after three state changes

Muestra `currentState` como nulo, y entonces inmediatamente se ve que se pasa al estado de naranja, y, finalmente, la aplicación está completamente en el estado de naranja, por lo que el evento de cambio se distribuye.

En este momento la aplicación está en el estado de naranja y todos los eventos estatales se hacen al hacer clic en el botón cambiar a negro, observe que el primer evento es un evento que cambia y se nota que el `CurrentState` es de color naranja. A continuación se ve que el estado naranja se sale, y entra en el estado negro, redondeando si fuera poco con un cambio final que demuestra que el nuevo valor para `currentState` es negro. Después de un tercer botón de clic se ve lo mismo pero de sentido opuesto, el cambio de negro a naranja. No vas a hacer nada dentro de su evento manipuladores, pero como usted puede ver, usted tiene conocimiento, desde la perspectiva del código, de todos los niveles de su aplicación cuando pasan a través de los cambios de estado. Usted puede construir en esto y lo utilizan para guardar el estado actual del usuario en un objeto compartido, base de datos, y así sucesivamente, por lo que la próxima vez que abran la aplicación que volver al estado normal.

Los eventos estatales no son diferentes de cualquier otro evento. Ellos proporcionan una ventana al proceso de estado de cambio y permitir un control total sobre su aplicación durante el estado de los cambios. Puede usar los eventos para almacenar último estado de la vista de modo usuarios que regresan pueden comenzar en el último lugar o desencadenar una nueva llamada de datos para tomar nueva información. Sus estados habilitados están listos para su publicación ahora. En la siguiente sección se toma su recién descubierta conocimiento y construir una aplicación real simulado.

14.3 Llevar juntos

Hemos cubierto todas las funciones de los estados: cómo cambiar las propiedades, controladores de eventos, grupos de estado, y gestión de eventos de estado. Vamos a explorar cómo se pueden utilizar los estados en un aplicación real. Listing 14.10 muestra el código de la aplicación.

Listing 14.10 Real-world state example

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/halo"
               xmlns:views="views.*"
               applicationComplete="init()">

  <s:layout>
    <s:VerticalLayout paddingLeft="20" paddingTop="20" />
  </s:layout>
  <s:states>
    <s:State name="login" stateGroups="loggedOut" />
    <s:State name="computers" stateGroups="loggedIn" />
    <s:State name="info" stateGroups="loggedIn" />
    <s:State name="tv" stateGroups="loggedIn" />
  </s:states>
  <s:Panel includeIn="loggedOut" title.login="Get in there!">
    <s:layout>
      <s:VerticalLayout />
    </s:layout>
    <mx:Form>
      <mx:FormItem label="Username">
        <s:TextInput />
      </mx:FormItem>
      <mx:FormItem label="Password">
        <s:TextInput />
      </mx:FormItem>
    </mx:Form>
  </s:Panel>
  <s:HGroup includeIn="loggedIn">
    <s:Button label="Login" click="currentState='computers'"/>
  </s:HGroup>
  <mx:ViewStack id="contentStack" includeIn="loggedIn">
    <views:ComputersView label="Computers" />
    <views:InfoForm label="Info" />
    <views:TVView label="TVs" />
  </mx:ViewStack>
</s:Application>
```

State declaration for
loggedOut group

State declarations
for loggedIn
group

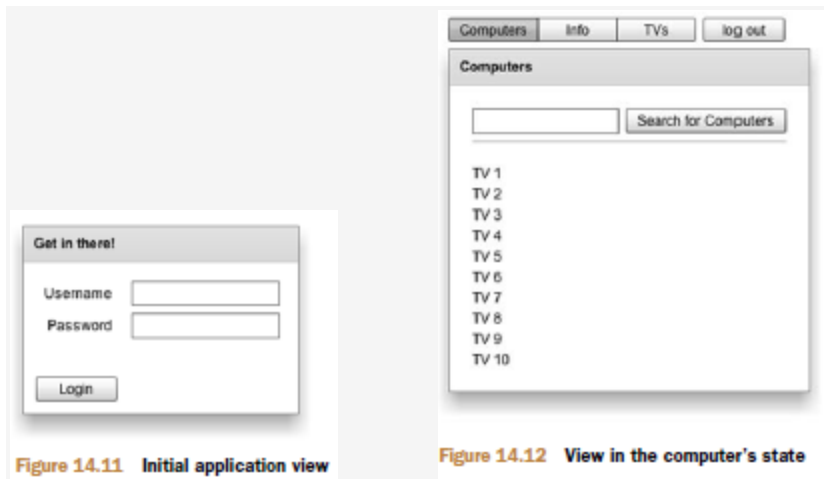
Login view

Application menu

ViewStack
showing content

Los estados son definidos en grupos, en función del estado de la “sesión” del usuario actual, logged in o logged out. Por brevedad, estás fingiendo una cuenta real, así que el botón Login cambia al estado computers, que muestra al usuario como conectado. Una vez que estés conectado, se muestra el menú y el ViewStack. Esto proporciona tres botones para el ViewStack y un botón Log out, que le envía de nuevo al estado login.

La figura 14.11 muestra la vista inicial de la aplicación, y figura 14.12 muestra el estado logged in.



Esto sigue siendo un ejemplo sencillo, pero se puede ver ahora dónde se puede relacionar en la autenticación de usuario, cambiar estados, y ver la aplicación completa con el usuario autenticado. La sintaxis es enormemente más fácil que la de Flex 3, ya que lleva a un código mucho más limpio, más ágil, y más legible. Ahora estás equipado y listo para utilizar estados. Felicitaciones al comenzar a utilizar MXML al máximo!

14.4 Resumen

Los Estados son de gran valor para su caja de herramientas de desarrollo. Usted puede utilizar los estados para manipular propiedades de los componentes, configurar los manejadores clic específicos, añadir y eliminar objetos de la pantalla, controlar cuando los elementos se crean y/o eliminan de la lista de visualización, mover objetos entre los padres, y controlar cada fase de los estados a través de eventos. En el siguiente capítulo aprenderá cómo integrar los servicios de datos en su aplicación.

15 - Trabajar con los servicios de datos.

Este capítulo trata de:

- Desarrollo centrado en datos con Flash Builder.
- Conexión a web services.
- Utilización de HTTPService y componentes WebService.
- Entender Action Message Format (AMF).
- Comunicación con Java EE utilizando BlazeDS.
- Comunicación ColdFusion.
- Comunicación con PHP vía Zend_Amf.
- Configuración del entorno de desarrollo para un flujo de trabajo cliente/servidor sin problemas.

Comunicación cliente/servidor RIA: la integración de una aplicación cliente RIA con su correspondiente aplicación de servicios del lado del servidor. Comprender el funcionamiento interno de cómo una aplicación Flex se comunica con los servicios de datos que proporcionará los conocimientos necesarios para poder conectar una aplicación Flex o AIR para cualquier tecnología de servidor, incluso si usted no sabe el lenguaje de servidor que se utiliza .

15.1 Acceso a los datos del lado del servidor.

Arquitectos RIA a menudo se encuentran desconcertados ante el dilema de elegir una infraestructura de servicios web. El conocimiento de la tecnología que está disponible en esta área es fundamental, porque la decisión equivocada podría ser catastrófico. Tabla 15.1 es una visión general de alto nivel de los servicios que están a su disposición y se puede utilizar como referencia a la hora de tomar decisiones de diseño de infraestructura.

Table 15.1 Web service protocol matrix

Communication Server support	Application	Benefits
HTTP (includes REST and RPC hybrids)		
- All	Simple widget-based applications; speed and real-time UI updates aren't required.	Easy implementation via the <code>HTTPService</code> object; RPC hybrid protocols can be invoked using <code>RemoteObject</code> .
SOAP/WSDL		
- All	Data aggregation from external web services.	Easy implementation; pull data from multiple outside resources regardless of platform.
AMF		
- BlazeDS & LiveCycle Data Services (LCDS): Java, - .NET/Zend: PHP - AMFPHP: PHP - WebOrb: .NET, Ruby, PHP	Approaching enterprise level; speed is important; data is usually pulled from server by polling.	Binary data compression makes communications 12 times faster; strong data typing; multiplatform support.
RTMP		
- LiveCycle Data Services (LCDS), - Flash Media Server (FMS)	Enterprise level, messaging, instantaneous UI updates; data can be pushed to the client; streaming media content; data-intensive RIAs.	Integrates into existing J2EE infrastructure; document management, rapid data transfer, clustering, data tracking, syncing, paging, and conflict resolution.
Flash Remoting		
- Native to ColdFusion	Robust, enterprise platform for client/server Flex communications; native.	Seamless integration with the Flash platform; removes the need for an intermediate code library to do data type mapping and data serialization.
JSON		
- All (JavaScript data objects are serialized and transferred in binary form.)	AIR applications that use AJAX or Flex applications that use the <code>ExternalInterface</code> API.	Easy implementation with the <code>HTTPService</code> object; part of the <code>AS3CoreLib</code> library.

La Tabla 15.1 es una matriz de protocolos básicos, que proporciona un punto de partida para saber cuándo usar cada una de las tecnologías disponibles para la comunicación de datos Flex junto con los beneficios que se pueden obtener para cada uno de ellos.

15.1.1 Usando del objeto `HTTPService`.

Dado que el objeto `HTTPService` utiliza el mismo paradigma de request-response que el navegador de Internet cuando muestra contenido de la web, se puede utilizar para invocar operaciones sobre cualquier tecnología de servidor con GET y POST. No requiere una biblioteca de código intermedio para la serialización a nivel de socket y el análisis como otros protocolos de comunicación de datos hacen. El listing 15.1 muestra cómo el objeto `HTTPService` se declara en MXML por lo que se puede invocar después de ejecutar una consulta de búsqueda en el motor de búsqueda de Yahoo!.

Listing 15.1 The HTTPService object declared in MXML notation

```
<mx:HTTPService
  id="yahooHTTPService"
  url="http://search.yahooapis.com/WebSearchService/V1/webSearch"
  method="GET"
  makeObjectsBindable="true"
  result="responseHandler(event)"
  fault="httpFaultHandler(event)"
  showBusyCursor="true">
</mx:HTTPService>
```

← Request can be GET or POST
← Makes result object bindable
← Declare the method to handle the result
← Declare a method to handle a fault
← Show busy cursor while in progress

Antes que el servicio sea invocado, se establece un objeto request con los argumentos necesarios y se pueden enviar con la llamada al servicio.

Listing 15.2 Calling the HTTPService from ActionScript

```
<fx:Script>
  <![CDATA[
    import mx.rpc.events.ResultEvent;
    import mx.rpc.events.FaultEvent;

    public function sendHttpRequest():void {
      var requestObj:Object = new Object();
      requestObj.appid = new String("YahooDemo");
      requestObj.query =
        new String("Flex in Action");
      requestObj.results = new int(2);
      yahooHTTPService.request = requestObj;
      yahooHTTPService.send();
    }

    private function
      responseHandler(e:ResultEvent):void {
      trace("Received a result: " + e.result);
    }

    private function
      httpFaultHandler(e:FaultEvent):void {
      trace("Received a Fault: " + e.message);
    }
  ]]>
</fx:Script>
```

1 Generic object created to send with request
2 Service variables passed into object
3 Generic object passed to property
4 HTTPService.send() method called
5 Response handler
6 Fault handler

Como se muestra en el listing 15.2, la función `sendHttpRequest()` invoca al objeto `HTTPService`. Antes de que eso ocurra, sin embargo, se crea un objeto genérico denominado `requestObj` (1) y se le pasa las variables `appid`, `query`, y `results` (2), que el service Yahoo! esperaba encontrar en el objeto genérico cuando recibe la solicitud. El `requestObj` se pasa luego en la propiedad `HTTPService.request` (3), y, finalmente, se llama al método `HTTPService.send()` (4).

Además, las funciones `responseHandler` y `FaultHandler` (FG) se implementan para capturar el evento que se desencadena por la respuesta del servicio a la solicitud. Como se puede ver en la `Alert.show` que se llama en el cuerpo de la función de `responseHandler`, los datos que se devuelven desde el servidor pueden ser accedidos a través de la propiedad genérica `ResultEvent.result`. Cualquier tipo de datos puede ser transmitido a través de esta propiedad, incluso los objetos `value` (o los objetos `data transfer` si vienes del mundo de Java o .NET). Has aprendido todo sobre el objeto `HTTPService`, lo que hace de esta una excelente oportunidad para presentarte al objeto `WebService`, como se verá en la siguiente sección.

15.1.2 Consumir web services con el WebService component

Web Service Description Language (WSDL) es un formato estándar para la descripción de servicios web basados en SOAP y RPC. Este tipo de consumo de datos es ideal para aplicaciones mash-up-style, ya que muchos servicios en línea han adoptado el estándar Simple Object Access Protocol (SOAP), pero esta es en general la manera más lenta de transferencia de datos, ya que lleva una sobrecarga con ella. Por otro lado, SOAP es similar a HTTP en el que soporta casi todas las plataformas del lado del servidor.

La configuración de la integración de servicios web en Flex es fácil y toma sólo unas pocas líneas de código. Por ejemplo, si desea utilizar la etiqueta `<mx:WebService>` para conectar con el servicio Web de tiempo, se vería como el código en el siguiente listado.

Listing 15.3 Using the WebService component

```
<mx:WebService id="weatherService"           Declare the WSDL document location ①
    wsdl="http://www.webservice.net/WeatherForecast.asmx?WSDL"
    fault="wsdlFault(event)">
    <mx:operation name="GetWeatherByZipCode"
        result="weatherResponse(event)"
        fault="weatherFault(event)" />
    <mx:operation name="GetWeatherByPlaceName"
        result="weatherResponse(event)"
        fault="weatherFault(event)" />
</mx:WebService>
```

② Multiple operation declarations within a WebService

La etiqueta `<mx:WebService>` contiene la información necesaria para indicar la aplicación Flex en el documento WSDL (1). Cuando se inicializa la etiqueta `<mx:WebService>`, se analiza el documento WSDL y extrae la información que necesita para generar objetos de Flex con la que se puede interactuar con el servicio web.

La operación de etiquetas (2) define las distintas operaciones que se utilizan con este servicio web. En este caso, las operaciones `GetWeatherByZipCode` y `GetWeatherByPlaceName` están definidas y listas para que usted las pueda invocar en la misma manera como lo hizo antes con el objeto `HTTPService`. Cada operación tiene un controlador definido para hacer frente a la respuesta de la solicitud de servicio.

TIP En algunos casos, un servicio web que desea conectarse puede utilizar métodos que son palabras reservadas en Flex. En estas situaciones, se puede utilizar la función `WebService.getOperation("nameOfOperation")` para obtener un identificador para la operación.

Flex elimina la responsabilidad de traducir los paquetes SOAP XML en objetos utilizables mediante la abstracción de la funcionalidad para analizar los paquetes SOAP dentro de la clase `WebService`. Usted puede obtener el objeto de datos que se crea después de que se complete el análisis del objeto genérico `ResultEvent.result` después del análisis de los paquetes de datos se ha completado y el `ResultEvent` se disparó y capturado por su función de control de resultados.

Flex elimina la responsabilidad de traducir los paquetes SOAP XML en objetos utilizables mediante la abstracción de la funcionalidad para analizar los paquetes SOAP dentro de la clase `WebService`. Usted puede obtener el objeto de datos que se crea después de que se complete el análisis del objeto genérico `ResultEvent.result` después que el análisis de los paquetes de datos se ha completado y el `ResultEvent` se disparó y capturó por su función de control de resultados.

15.2 Action Message Format en acción.

Como se vio de nuevo en la tabla 15.1 al comienzo del capítulo, Action Message Format es un protocolo de comunicación sólida que se está convirtiendo rápidamente en el método preferido de comunicación entre los desarrolladores de RIA por su tecnología abierta, la velocidad y el soporte nativo para la máquina virtual de Flash .

Para mostrar cuán rápido AMF es, James Ward(Adobe técnico superior) escribió una aplicación Census RIA Benchmark. Él ha estado actualizando en el transcurso de los dos últimos años con nuevas características y ha incluido tantas formas de transferencia de datos como la comunidad ha sido capaz de tirarle. A pesar de la dura competencia de algunos desafíos serios de la comunidad, el protocolo AMF sigue siendo el campeón.

La figura 15.1 proporciona una captura de pantalla de la aplicación, que se puede encontrar en <http://www.jamesward.com/census/>.



Figure 15.1 James Ward's benchmark illustrates how impressive AMF is for data communication.

AMF le presenta una serie de opciones en lo que respecta a su aplicación, pero las opciones son por lo general reducidas por su elección de tecnología del lado del servidor. En las siguientes secciones, vamos a hacer un breve resumen de las tecnologías disponibles en el momento de escribir este artículo.

15.2.1 Open-source AMF.

En diciembre de 2007, Adobe anunció que AMF se convertiría en un protocolo abierto, que inmediatamente llevó al desarrollo de una serie de librerías de código, muchos de los cuales imitaba las capacidades de transferencia de datos de ColdFusion a través de Flash Remoting.

La apertura del protocolo fue la frutilla del pastel para la mayoría de los desarrolladores de la plataforma Flash. Ya era más rápido que cualquier otra cosa, y su apoyo es nativo de la máquina virtual de Flash. El número de bibliotecas de código fuente abierto que se desarrollaron poco después es prueba de ello.

Conocer las herramientas que están disponibles para la comunicación de datos con AMF es simple.

15.2.2 AMF con PHP.

El proyecto AMFPHP código abierto se inició en enero de 2008, justo después que AMF fuera de código abierto. Afortunadamente, Adobe formó una alianza estratégica con Zend Technologies en Q308, y fue desarrollado el módulo Zend_Amf e integrado en el framework de Zend PHP.

El módulo Zend_Amf es fácil de trabajar y actualmente es la solución ideal para la comunicación de datos entre Flex y PHP, especialmente si usted está utilizando Flash Builder para el desarrollo de aplicaciones Flex. Una de las grandes novedades de Flash Builder 4 es su capacidad para optimizar sus flujos de trabajo mediante la automatización del proceso de conectar los servicios de datos, como se verá en la sección 15.3.

15.2.3 AMF y ColdFusion.

En el momento de escribir esto, la versión beta de ColdFusion 9 se acaba de publicar, junto con la herramienta de desarrollo ColdFusion primera jamás realizada por Adobe, llamada CFBuilder. No es de extrañar, AMF es más efectiva cuando se trabaja con ColdFusion, ya que tiene soporte nativo para el formato AMF. Esto significa que cuando se ejecuta una aplicación de Flash plataforma RIA en el cliente con los servicios de datos de ColdFusion en el servidor, tienes soporte nativo para el mecanismo más rápido disponible de transferencia de datos en el cliente y el servidor.

El resultado es una aplicación rica de Internet que funciona como si se está ejecutando nativa en el escritorio local. Más específicamente, las cantidades masivas de datos pueden ser capturados, se filtra, organiza, procesa, y se muestran en un resumen gráfico legible en menos de un segundo. Sólo era hace unos años que este tipo de consulta se llevará a cabo por un técnico de investigación, que luego de levantarse e ir a comer y volver 45 minutos más tarde encontraría que la consulta no era completa todavía.

15.2.4 BlazeDS.

Adobe código abierto de la especificación AMF en conjunto con el lanzamiento de BlazeDS, una aplicación Java de código abierto que se puede utilizar para la integración de Java y Flex a través de AMF.

Con BlazeDS, los desarrolladores pueden invocar métodos de objetos preexistentes planos viejos Java (POJOs), Servicios de primavera, EJB y otras implementaciones de Java Enterprise System. Blaze también puede ser enganchado en JMS y Hibernate para aplicaciones que requieren mensajería. Servidores de aplicaciones Java compatibles con BlazeDS incluye Tomcat, WebSphere, WebLogic, JBoss, y ColdFusion.

15.2.5 LiveCycle Data Services.

BlazeDS era un descendiente de LiveCycle Data Services. Es lógico afirmar que BlazeDS es el hermanito de LCDS. El mercado objetivo de LCDS está fundamentalmente a gran escala en entornos empresariales que consisten en grandes granjas de servidores. Teniendo en cuenta el costo de una sola licencia LCDS, no es de extrañar que la mayoría de las pequeñas empresas y desarrolladores emprendedores view LCDS estén fuera de su alcance. Al igual que con BlazeDS, LCDS es una aplicación basada en Java de la AMF y ofrece ventajas adicionales que conducen a las necesidades de la empresa. El costo suele ser justificado por el nivel de soporte que viene con el paquete.

15.2.6 Tecnologías adicionales.

Otras tecnologías que han surgido en los últimos años incluyen WebORB, AMF.NET, AMFPHP y RubyAMF. Pero los tres que se destacan entre el resto en la arena RIA son Zend_Amf, ColdFusion Remoting y BlazeDS para Java EE. Nos centraremos en las tecnologías para el resto de este capítulo.

Ahora estás armado con el conocimiento de fondo que tendrá que embarcarse en su próxima misión: aprender a crear aplicaciones centradas en datos AMF con Flash Builder.

15.3 Creación de aplicaciones Flex centradas en datos con Flash Builder

El desarrollo centrado en los datos (DCD) que dispone Flash Builder se creó para ahorrar una cantidad significativa de tiempo en el desarrollo de aplicaciones Flex basadas en datos.

Lo primero a destacar con respecto a las características de desarrollo centradas en datos incluidos en Flash Builder 4 es que el flujo de trabajo en general sigue siendo el mismo, independientemente de la tecnología de servidor que se utiliza. Más adelante se van a señalar las diferencias en el flujo de trabajo que si dependen de la tecnología del servidor.

El tiempo de desarrollo se reduce mediante la implementación de una serie de asistentes que guían a través del proceso de conexión con el código del lado del servidor y transfieren los datos fuertemente tipados. Mientras se avanza con los asistentes, se genera el código en segundo plano. El siguiente ejemplo muestra cómo se puede realmente simplificar el flujo de trabajo para la conexión a los servicios de datos sin salir de la vista de diseño del IDE Flash Builder!

15.3.1 Creación de un entorno adecuado

En el desarrollo de aplicaciones RIA, el entorno de desarrollo integrado (IDE) debe reflejar la plataforma tanto del cliente como del servidor en que se está trabajando.

La tabla 15.2 proporciona una lista de recomendaciones sobre cómo se puede configurar el entorno en base a la elección de la plataforma del servidor. El concepto de la matriz IDE se muestra en la tabla 15.2 es el de ser capaz de hacer tanto de cliente como el desarrollo del lado del servidor sin salir del entorno Eclipse. Esto hará más eficiente el flujo de trabajo y permitirá ahorrar mucho tiempo.

Table 15.2 Eclipse IDE configuration matrix for data-driven Flex 4 RIA development

PHP	Eclipse PDT + Flash Builder 4 plug-in	Zend Studio + Flash Builder 4 plug-in
J2EE/Blaze/LCDS	Eclipse for Java + Flash Builder 4 plug-in	Eclipse for Java EE + Flash Builder plug-in
ColdFusion	CFEclipse + Flash Builder 4 plug-in	CFBuilder + Flash Builder 4 plug-in
WSDL	Flash Builder 4 + WDT Eclipse plug-in	WDT Eclipse + Flash Builder 4 plug-in
.NET	Flash Builder 4 + WDT Eclipse plug-in	WDT Eclipse + Flash Builder 4 plug-in

Con Flex y Flash Builder 4, no hay necesidad de sentirse abrumado por la elección de protocolo para las comunicaciones de datos. Flash Builder 4 se encarga de la mayor parte de esto en segundo plano cuando se utilizan los asistentes de DCD. Trata de usar Action Message Format para comunicaciones de datos siempre que sea posible, debido a su velocidad de transferencia.

15.3.2 Establecimiento de conexión con el servidor

Debido a que la estructura de Flex se basa en una arquitectura Modelo-Vista-Controlador, por lo general es de interés mantener la coherencia con la metodología MVC en la aplicación también. Esta sección se centrará en el uso del Flash Builder GUI para generar el código para la capa de modelo de la aplicación. Pero primero es necesario establecer una conexión con la capa de aplicación de servidor.

TIP El flujo de trabajo del desarrollo centrado en datos integrado en el IDE de Flash Builder 4 es casi idéntico entre la integración con PHP y la integración con los servicios de ColdFusion.

Hay dos escenarios para la conexión a un servicio de datos: Ya sea que se está tratando de conectar a un proyecto que ya existe, o se va a empezar con uno nuevo. En el primer escenario, se tiene un proyecto que desea conectarse a un servicio de datos. Esto se hace fácilmente seleccionando Connect to Data/Service link mediante el panel Data/Services en Flash Builder, como se muestra en la figura 15.2.

En el segundo escenario, se puede establecer la conexión con el servidor durante la creación de un nuevo proyecto de Flash Builder. Como se muestra en la figura 15.4, el Asistente para configuración de proyectos incluye un menú desplegable que le permite seleccionar un servidor de aplicaciones para configuración durante el proceso de configuración del proyecto. Las opciones son Ninguno / Otros, NET, J2EE, ColdFusion y PHP.

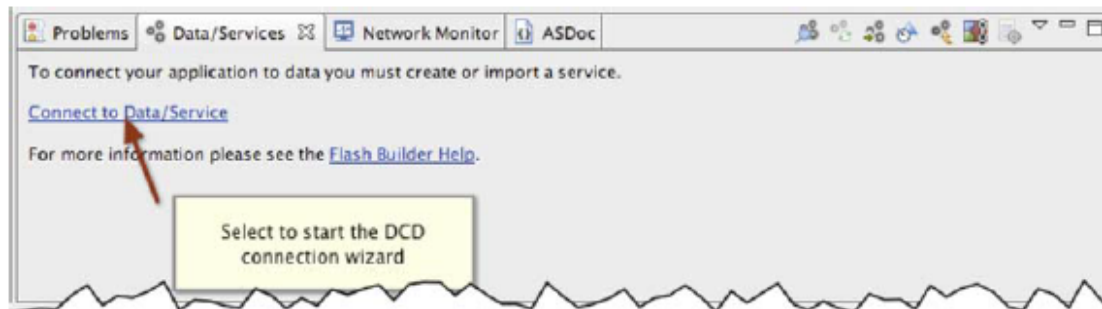


Figura 15.2 Seleccione Connect to Data/Service desde el panel Data /Service para iniciar el asistente de DCD y se desplegará la ventana de la figura 15.3, que muestra todos los tipos de servicios que se pueden conectar.

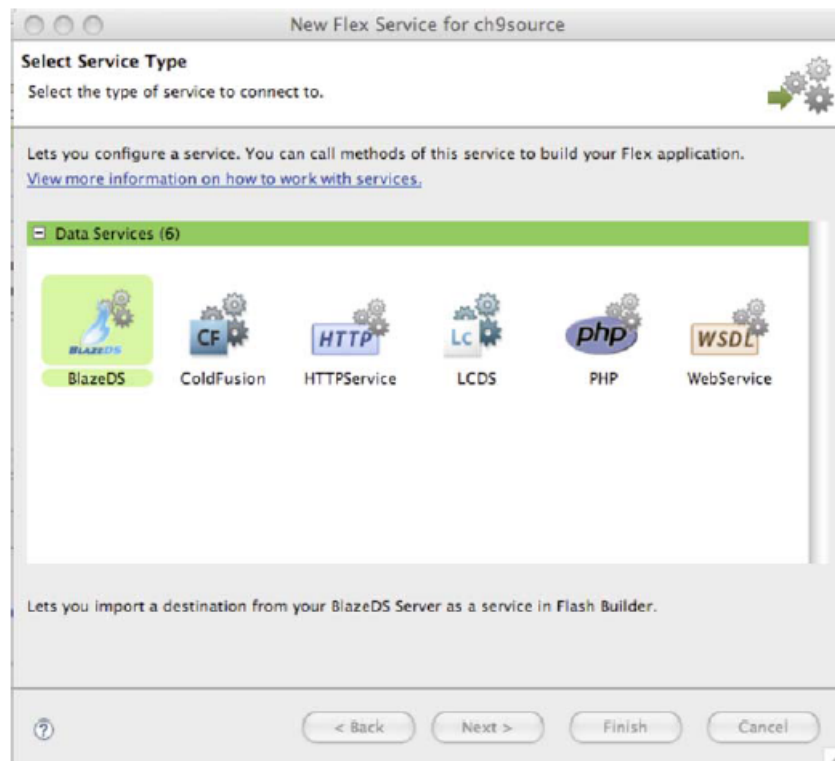


Figura 15.3 Seleccione de la lista de tipos de servicios disponibles.

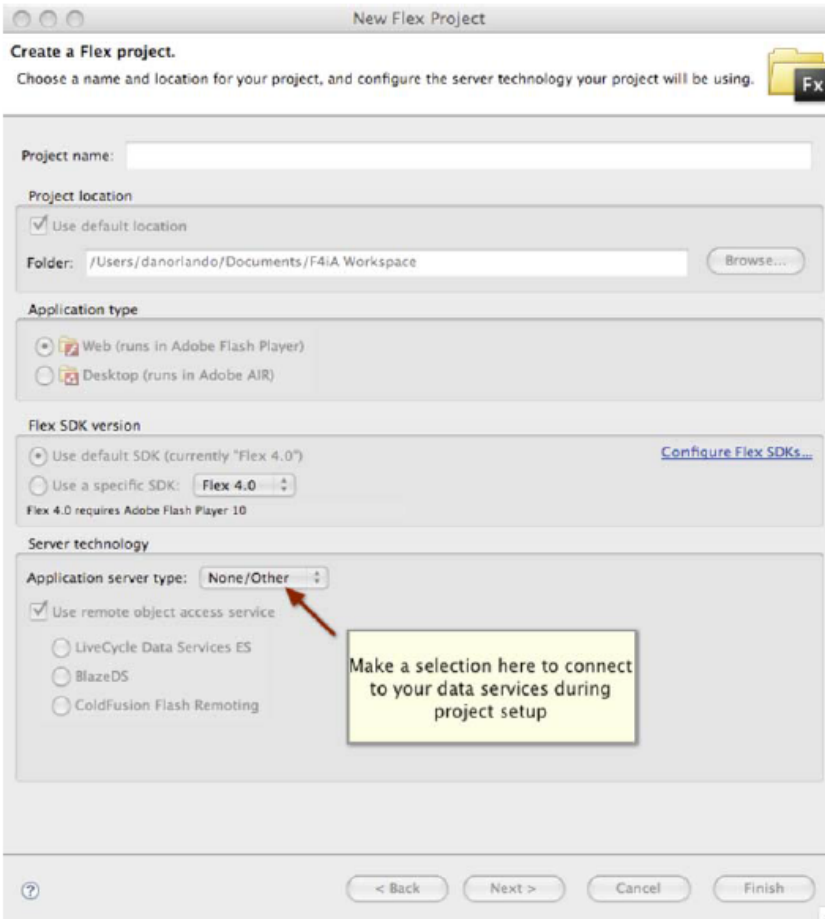


Figura 15.4 Use el menú desplegable para configurar el servidor de aplicaciones al crear un nuevo proyecto en Flash Builder.

Independientemente del escenario, se va a terminar en el mismo lugar, el cuadro de diálogo de configuración del servidor. La figura 15.5 muestra la ventana de diálogo de configuración de servidor después de seleccionado PHP como el tipo de servidor de aplicaciones.

Independientemente del tipo de servidor, Flash Builder estará preocupado por dos cosas:

- 1 El URI al que debe conectarse.
- 2 Validar que es capaz de realizar la conexión con éxito.

Después de introducir los parámetros necesarios en los campos de configuración, el botón Validar configuración quedará activado. El asistente no le permitirá continuar hasta que se valide correctamente la configuración del servidor. Se sabrá cuando la validación es correcta, porque se verá un mensaje en la parte superior de la ventana, como se muestra en la figura 15.5.

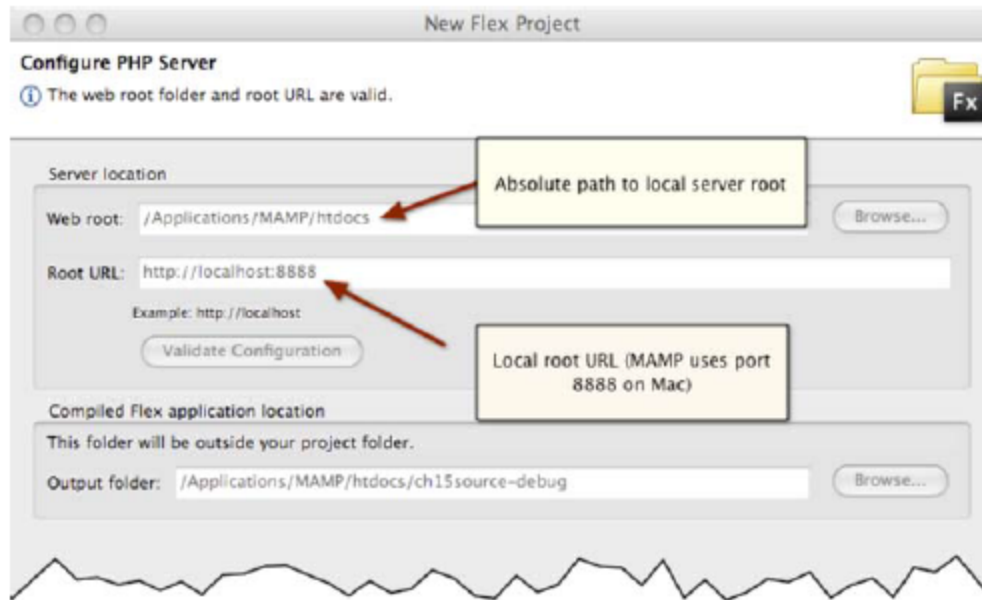


Figura 15.5 Configuración de un servidor local PHP DCD en Flash Builder.

NOTA: Si se está trabajando con PHP en el servidor, y Flash Builder ve que usted todavía no tiene instalado el Zend Framework, se le preguntará si desea que se instale. Esto se hace para que se pueda tomar ventaja de los beneficios de la AMF a través del módulo Zend_Amf. Asegúrese de seleccionar Sí si se le despliega este mensaje.

Generación de Servicios

Una de las mejores características de DCD en Flash Builder 4 es la capacidad de la aplicación para generar los servicios básicos con las operaciones CRUD estándar para una tabla de base de datos específica. También incluye otras operaciones útiles tales como la recuperación de conjuntos de resultados paginados (20 a la vez, por ejemplo).

Si ya está creada la base de datos, Flash Builder puede leer el esquema de una tabla específica y generar todo el código que necesario para generar un servicio básico a partir de la misma. Si la tabla contiene muchos campos, la cantidad de tiempo que esto puede ahorrar es incommensurable.

El proceso de generación de un stub de servicio comienza seleccionando el Click Here para generar un vínculo de muestra bajo el campo Ubicación PHP, como se muestra en la figura 15.6. Si el código de su servicio ya se ha escrito, sin embargo, puede utilizar esta ventana para especificar el nombre y ubicación del servicio para que pueda ser introspeccionado por Flash Builder.

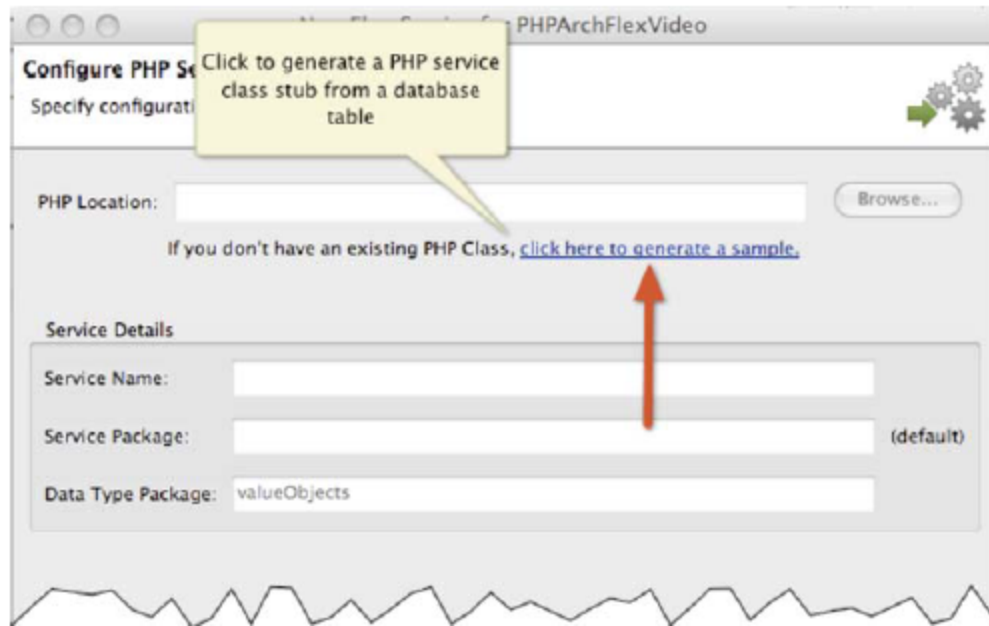


Figura 15.6 Servicios Stub pueden ser generados desde el cuadro de diálogo de configuración del servicio.

El cuadro de diálogo que aparece a continuación en la secuencia se muestra en la figura 15.7. Para sacar el máximo provecho de las características integradas DCD, asegúrese de haber creado primero una base de datos y una tabla con los campos dentro de su base de datos. A continuación, asegúrese de que Generate from Database ya está seleccionado cuando se abra la ventana del generador de servicio de ejemplo. Rellene el resto de los parámetros de acuerdo a la configuración de base de datos y haga clic en el botón Conectar con base de datos.

Suponiendo que todos los parámetros de entrada son correctos y Flash Builder se conectó a la base de datos que se ha especificado, se dará cuenta de que, de repente, el menú desplegable de la tabla se rellena con todas las tablas de la base de datos. Lo siguiente que hay que hacer es seleccionar la tabla que desea utilizar para generar el stub de servicio.

Detalle de lo sucedido durante el proceso:

- Se agregó un paquete en la carpeta src.
- Apareció una carpeta services en la carpeta libs.
- Se generó la clase servicio y se abrió en el panel de edición principal del IDE de Eclipse o el editor por defecto en su sistema para ese tipo de archivo.
- La clase fue introspeccionada en Flash Builder, y todos sus métodos se muestra en el panel Data/Service.

- Una clase abstracta fue creada en el proyecto que contiene métodos para llamar fácilmente cada una de las operaciones del servicio.
- También se creó una clase vacía que hace la llamada a la superclase (abstract). La clase vacía es donde se debe colocar el código personalizado que se pueda necesitar.

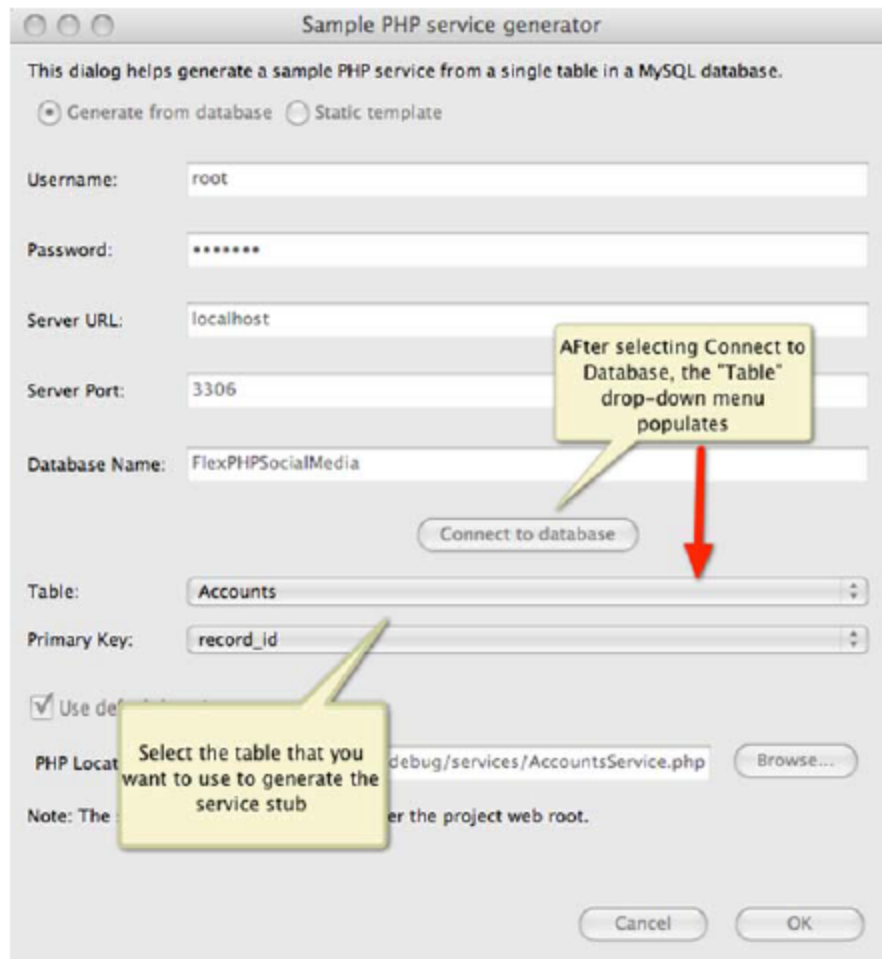


Figura 15.7 Los nombres de las tablas de base de datos aparecen en un menú desplegable después de hacer click en el botón Conectar con base de datos.

Si está trabajando en la versión completa de Flash Builder sin haber instalado el editor necesario para el código del servicio en el IDE, Flash Builder abre el servicio generado con la aplicación por defecto

para ese tipo de archivo (probablemente Adobe Dreamweaver, si está instalado en la máquina local).

MIGRACIÓN TIP Lo importante es entender que el viejo paradigma que se ha utilizado con las versiones anteriores del SDK de Flex-donde los objetos de valor eran creados en el servidor para que corresponda con los objetos de valor en el lado del cliente-ha cambiado significativamente. Más concretamente, no es necesario crear un objeto PHP fuertemente tipado que se corresponde con sus objetos de ActionScript). Para los servicios básicos CRUD, se necesita poca codificación en el servicio PHP autogenerado. Configurar los parámetros de conexión de base de datos y nombres de tablas en consecuencia.

Ahora que ya ha generado un stub de servicio, es el momento de configurar el envío y retorno de tipos. Se ha realizado todo esto sin salir de la vista de diseño de Flash Builder.

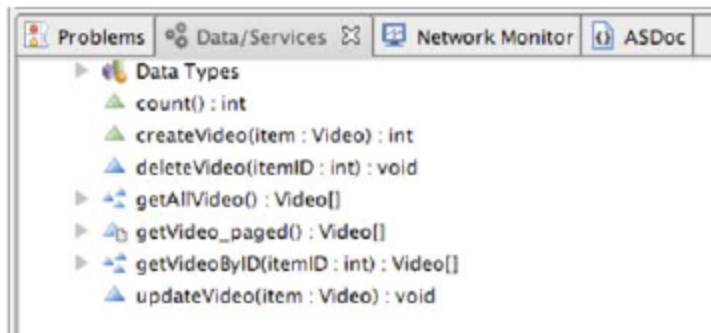


Figura 15.8 Flash Builder establece automáticamente el envío y retorno tipos de datos al servicio de la introspección.

Configuración de los tipos de datos de envío y retorno

Antes de empezar a configurar los tipos de datos manualmente, mira las tipificaciones de datos en el panel Data / Services. Si se establece correctamente el tipo en los campos de la base de datos, Flash Builder probablemente ya establecerá sus tipos de datos, como lo hizo para el proyecto que se muestra en la figura 15.8. Otro increíble ahorro de tiempo.

La aplicación Flash debe saber qué esperar de los tipos de objetos de datos que se envían y se retornan en cada operación. Para configurar los tipos de datos, puede hacer clic-derecho en una operación desde el panel Data / Services y luego hacer clic en Configurar tipo de retorno, o

seleccionar el método y luego hacer clic en el icono configure send / return type de la barra de herramientas del panel Data / Services. A continuación, debería ver una ventana que se parece a la de la figura 15.9.

Como se mencionó anteriormente, ya no hay que codificar más los propios objetos de valor. Se puede generar automáticamente el código necesario cuando se invoca el método `getAllItems()`. En el pasado, cuando un conjunto de objetos fuertemente tipado era retornado como respuesta del servidor, era una práctica habitual escribir los datos declarados como `ArrayCollection` en la parte superior de la clase y luego declarar en la lista o grid de datos la variable del tipo `ArrayCollection`. Todavía se pueden hacer las cosas de esta manera, si se quiere, pero vale la pena dejar que Flash Builder maneje esto porque hace el flujo de trabajo mucho más rápido. La figura 15.10 muestra cómo habilitar la detección automática del tipo de retorno de datos.

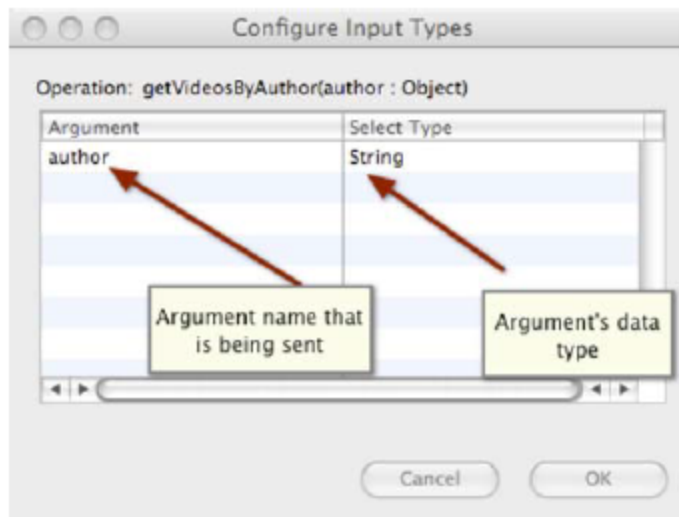


Figura 15.9 En la ventana Configure Input Types, puede configurar los parámetros que se van a enviar con la solicitud cuando se invoca la operación.

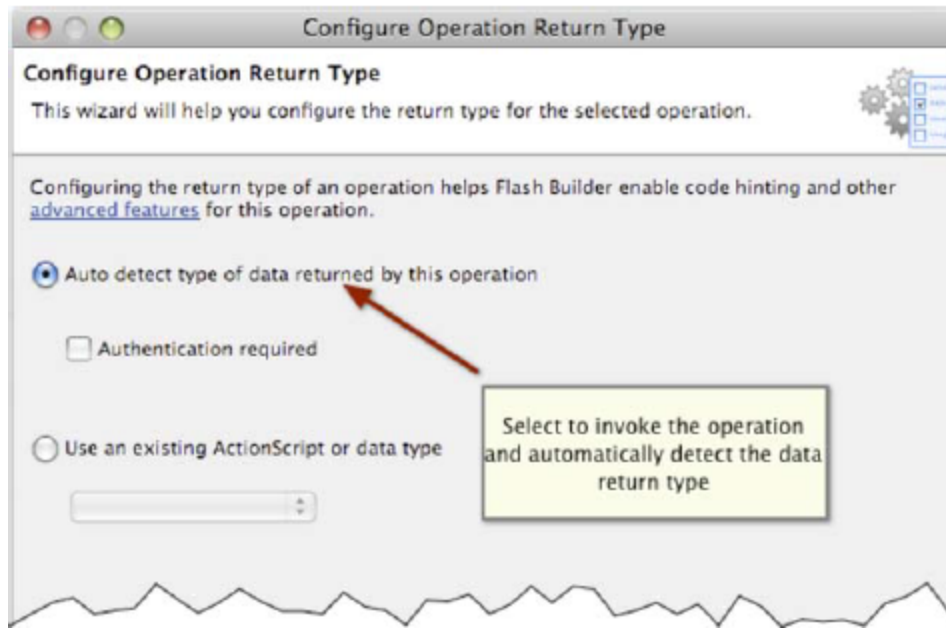


Figura 15.10 Los tipos de datos de retorno pueden ser detectados automáticamente por Flash Builder al invocar la operación.

Los objetos de valor que Flash Builder genera son más como objetos de valor, ya que hacen mucho más que envolver un grupo de valores de datos fuertemente tipados entre el cliente y el servidor. Por ejemplo, el conjunto de tipos de cambio de `getAllItems()` es un objeto de usuario, aunque se trata de una colección de objetos de usuario. Esto se debe a que los controladores de resultados generados automáticamente son lo suficientemente inteligentes para saber la diferencia entre un objeto de usuario que se devuelve en comparación con una colección de objetos de usuario. No hay que escribir el resultado en un `ArrayCollection` y pasar por toda la escritura, el proceso de unión, y objectmapping como se hubiera hecho con el viejo Flex Builder. Cosas como la asignación de clase también hicieron el proceso más difícil. Por suerte, eso es todo en el pasado.

Hasta ahora se ha visto la creación de una aplicación centrada en los datos sin salir de la vista de diseño de Flash Builder IDE, que es bastante prolijo. Siguiendo con este tema, aprenderá cómo realizar drag y drop de datos en determinado momento. Pero antes, vamos a construir en el servicio de datos un conjunto de habilidades y una base de conocimiento, mostrando la forma de trabajar con algunas otras tecnologías de servidor disponibles. Es importante saber cómo trabajar con una amplia gama de tecnologías de servidor, porque nunca se sabe con lo que se va a trabajar del lado del servidor en un próximo proyecto.

15.4 Data-centric Flex con ColdFusion

Si ya eres un desarrollador de PHP y tienes curiosidad sobre ColdFusion, Flash Builder 4 hace que sea especialmente fácil empezar. Adobe finalmente ha añadido una poderosa herramienta de desarrollo soportada específicamente para el desarrollo de ColdFusion a su paleta en el IDE, llamada CFBuilder. Desde el punto de vista de desarrollo de Flex, la cosa más fresca sobre CFBuilder es que, como Flash Builder, CFBuilder se basa en Eclipse.

Esto significa que CFBuilder se puede instalar directamente en el IDE de Flash Builder como un plug-in, como se muestra en la figura 15.11, para que pueda manejar tanto el lado del cliente y el desarrollo del servidor de las aplicaciones RIA sin salir de su IDE primario. Nunca ha habido un mejor momento para empezar a utilizar ColdFusion con Flex que ahora.

Para configurar un nuevo proyecto Flex para el uso con los servicios de ColdFusion, seleccione ColdFusion para el tipo de servidor de aplicaciones en la primera pantalla del Asistente para nueva configuración de proyectos y elija el botón de Flash Remoting. Se le pedirá entonces que establezca la configuración del servidor como lo hizo en el ejemplo anterior.

El propósito de la figura 15.12 es reiterar el punto de que el proceso de conseguir crear servicios de datos es generalmente el mismo, independientemente de la tecnología de servidor. Figura 15.12 debería parecer familiar porque es casi idéntica a la figura 15.5, donde se ha configurado y validado un servidor PHP. En este caso, sin embargo, está configurando el proyecto para el uso con los servicios de datos de ColdFusion.

Después de completar el Asistente para configuración de proyectos, seleccione Conectarse a Data/Service, como lo hizo antes. La siguiente ventana pop-up se verá como en la figura 15.13, donde le presentan la opción de establecer un servicio de ColdFusion (Flash Remoting, en este caso), HTTPService o WebService.

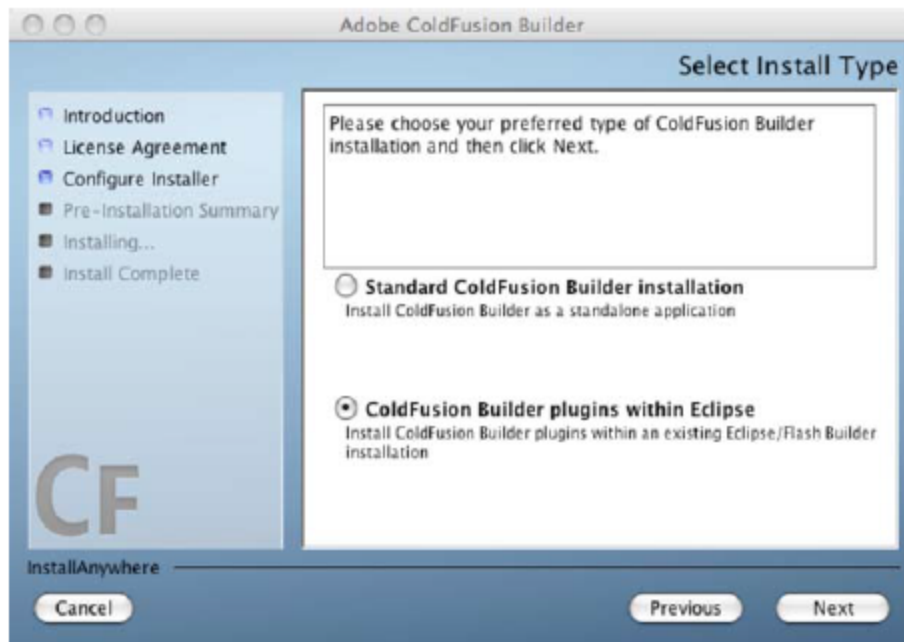


Figura 15.11 Adobe CFBuilder puede ser instalado como un plug-in en Flash Builder 4 IDE.

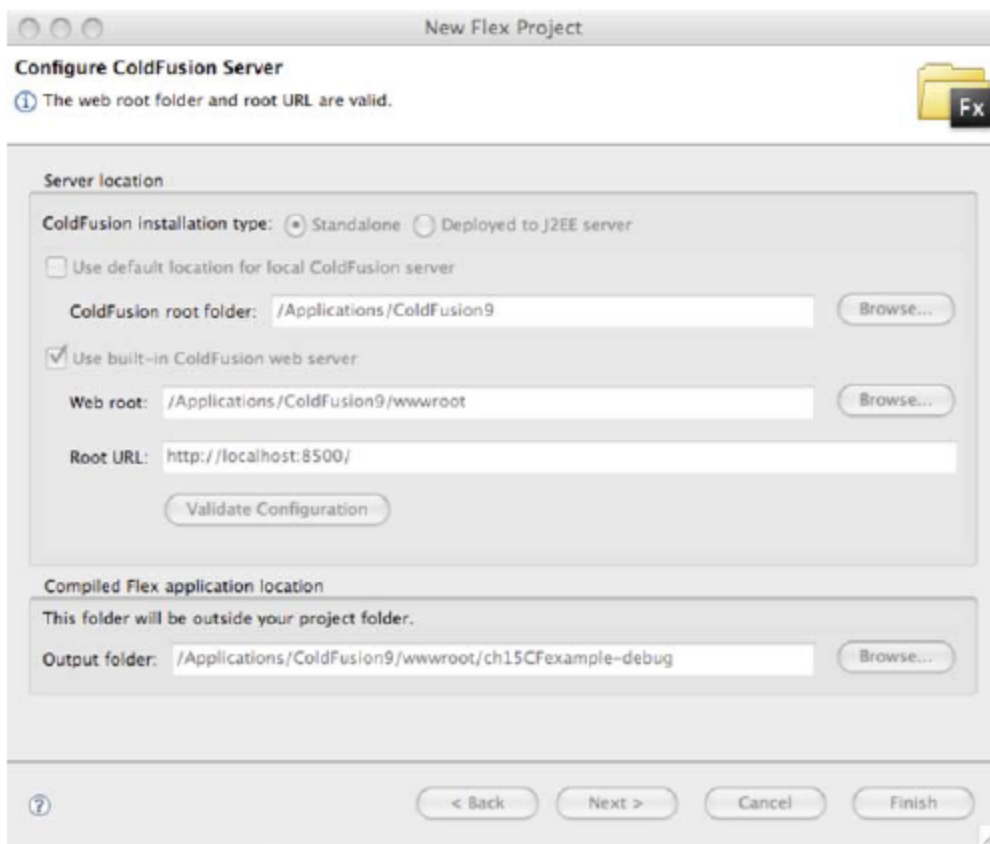


Figura 15.12 Configuración de un proyecto Flex para los servicios de datos de ColdFusion

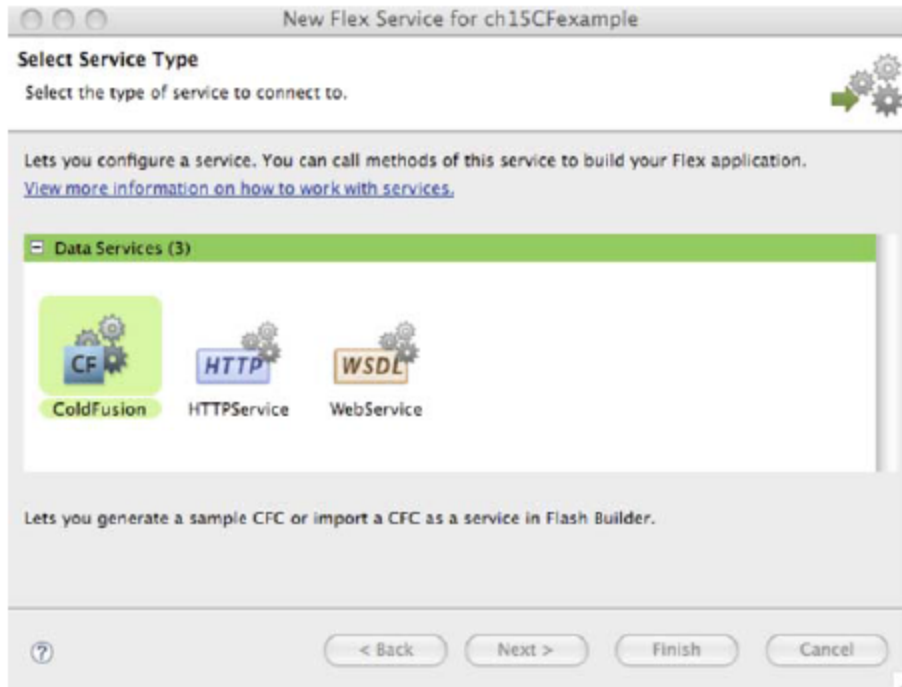


Figura 15.13 La selección de ColdFusion en este caso le permite tomar ventaja de built-in Flash Remoting.

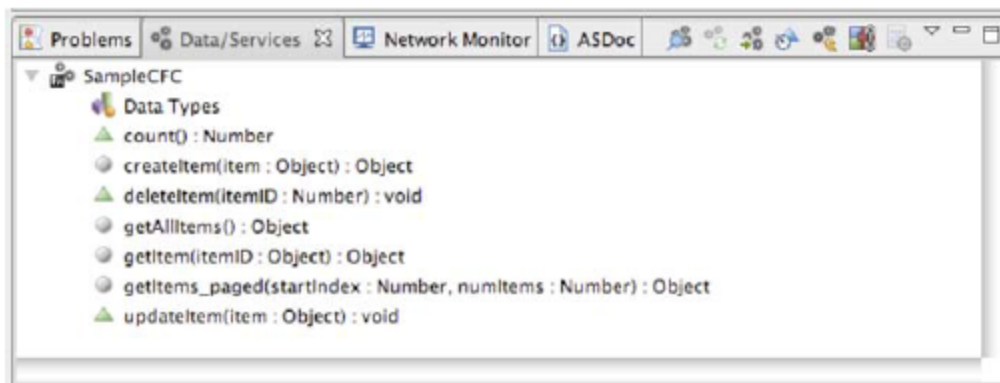


Figura 15.14 Las operaciones stub se muestran en la ventana Data/Services listo para la unión drag and-drop

El resto del proceso de configuración del servicio es el mismo que para el ejemplo anterior, y como se muestra en la figura 15.14, las operaciones que están disponibles una vez que se genera el stub de servicio también son las mismas.

Ahora que ha echado un vistazo al desarrollo centrado en los datos ColdFusion con Flash Builder, vamos a echar un vistazo rápido a los datos centrados en Java EE con BlazeDS antes de pasar a drag-and-drop data binding (enlace de datos).

15.5 Centrado en los datos Flex con Java EE y BlazeDS

Para el desarrollo de la aplicación Flex en el cliente, la creación de un proyecto Flex para su uso con aplicaciones web J2EE es tan simple como seleccionar J2EE en el menú desplegable Tipo de servidor de aplicaciones cuando aparezca la primera ventana del Asistente para nueva configuración de proyectos, como se muestra en la figura 15.15.

Después de seleccionar J2EE, se le ofrece la opción de utilizar LiveCycle Data Services o BlazeDS. Para este ejemplo, seleccione BlazeDS porque es poderoso, es de código abierto, y lo mejor de todo, sólo se tarda unos 15 minutos en llegar a funcionar con él.

Configuración de BlazeDS

El lanzamiento de Builds de BlazeDS vienen en tres opciones: Llave en mano, distribución binaria, y fuente. La forma más rápida de ponerse en marcha es el despliegue de la descarga de llave en mano. Las versiones de lanzamiento de BlazeDS están disponibles en el sitio web de código abierto de Adobe en <http://opensource.adobe.com/wiki/display/blazeds/Release+Builds>.

Además, Adobe evangelista Sujit Reddy G tiene un gran post sobre creación de BlazeDS en <http://sujitreddy.wordpress.com/2009/04/07/setting-up-blazeds/>.

La documentación BlazeDS que se proporciona en el sitio Web de Adobe de código abierto también incluye instrucciones paso a paso.

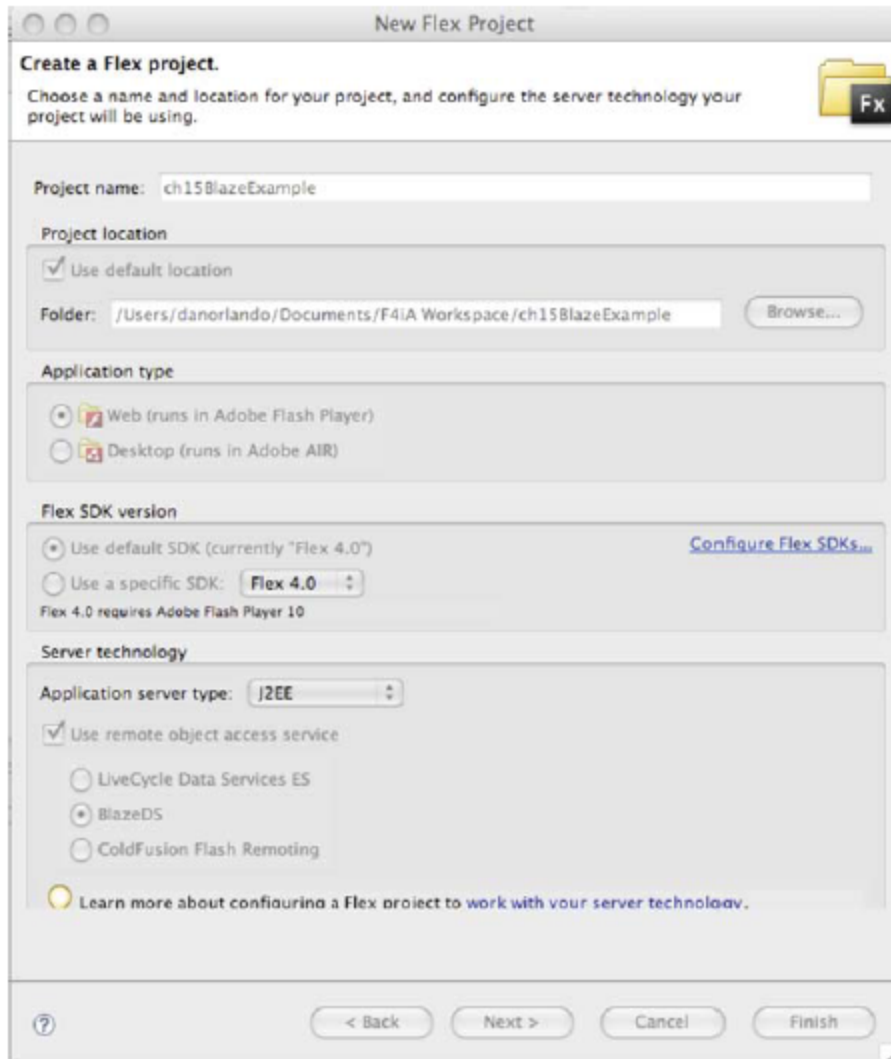


Figura 15.15 Creación de un nuevo proyecto Flex para su uso con J2EE y BlazeDS

15.6 Enlazar el modelo a la vista

En el patrón de diseño MVC, el modelo es la colección de datos que se recuperan del lado del servidor mediante la invocación de operaciones de un servicio, la vista es la visualización de dichos datos, y el controlador es el código responsable de la unión el modelo y la vista juntos. En teoría, esto

suenan muy bien, pero en la práctica, escribir todo el código puede ser tedioso y aburrido. Introduzca drag and drop en el enlace de datos.

15.6.1 Enlace de datos con drag and drop

Una de las mejores cosas acerca de Flash Builder 4 es la capacidad de arrastrar y soltar las operaciones del panel Data / Services sobre los componentes basados en listas en la vista de diseño, creando un enlace entre la respectiva operación y el componente sobre el que se soltó.

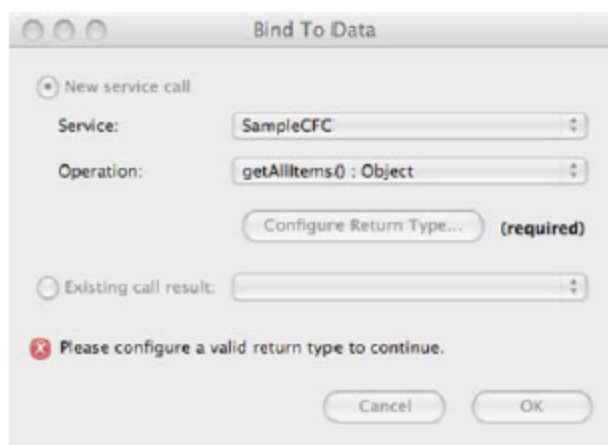


Figura 15.16 Si se muestra esta ventana, hay que volver atrás y configurar los tipos de retorno.

Tres simples pasos para el enlace de datos

Asegúrese de que está en la vista Diseño con el archivo principal de la aplicación XML seleccionado en la ventana principal.

1. En primer lugar, tomar un VDividedBox y arrastrarlo al escenario. Establecer los valores X e Y en cero, y establecer las propiedades height y width de 100%.
2. A continuación, tomar un componente DataGrid de la lista de componentes y arrastrarlo al escenario para que esté dentro de la VDividedBox. Establecer sus valores X e Y en 0 también. A continuación, establecer su ancho de 100% y una altura de hasta 30%.
3. A continuación seleccione el método getAllItems() en la ventana Data/ Services y arrástrelo sobre el dataGrid.

Si no establece el tipo de retorno de sus métodos, verá una ventana emergente que tiene una

aparición similar a la figura 15.16. De lo contrario, usted debe ver de inmediato los encabezados de columna cargada con los nombres de los campos, y el número de columnas debería cambiar en función del número de campos que tiene.

Una buena manera de identificar errores es volver e invocar los métodos a través de la configuración del tipo de retorno de cada operación. Las respuestas de error que vuelven son muy detalladas y por lo general dirán exactamente cuál es el problema y dónde ocurrió. El Monitor de red también será muy útil durante sus esfuerzos de experimentación.

15.6.2 Generación de un formulario principal de detalle

Una cosa es segura: el enlace de datos a través de drag and drop es maravilloso, pero la generación de un formulario principal de detalle sin tener que escribir una sola línea de código es aún más genial!

Para empezar, haga clic derecho en el componente DataGrid en la vista de diseño y seleccione Generar Detail Form. Como se demuestra en la figura 15.17, la ventana que se muestra debería tener seleccionada Master-Detail, así como el check box Form Editable seleccionado. Desmarque esta casilla si desea que los detalles del elemento seleccionado se muestren como campos de texto no editables.

A continuación, seleccione la casilla de verificación Hacer un nuevo servicio de llamada para obtener detalles. Asegúrese de que está seleccionado el servicio correcto y para el menú Operación, seleccione el método getObjectById (), como se muestra en la figura 15.17. A continuación, haga clic en Siguiente. La siguiente ventana debe ser similar a la figura 15.18.

La ventana Asignación de Control de Propiedades que se muestra (figura 15.18) le ofrece la oportunidad de anular la selección de los campos que no desea que se muestren en el formulario principal. También puede dejar un campo seleccionado y seleccionar Texto para el control, lo que significa que se mostrará, pero no será un elemento editable.

Un campo de identificador único es un buen ejemplo de este tipo de situación, que se ilustra en la figura 15.18, donde video_id es el identificador único para cada registro. Si usted ha estado siguiendo los pasos, junto con un proyecto propio, debería ser capaz de ejecutarlo en este momento con todas las características y funcionalidades disponibles. Todo esto lo hizo sin dejar en ningún momento la

vista de diseño de Falsh Builder.

Generate Form

Choose source

Select the source for which the form should be generated. Configure the form.

Generate Form for: Master-Detail

Component displaying master data: DataGrid

☒ Make form editable

Optional service call to get details

☒ Make a new service call to get details

Service: VideoService

Operation: getVideoByID(itemID : int) : Video[]

Configure Return Type...

? < Back Next > Cancel Finish

Figura 15.17 Asegúrese de estar usando la configuración correcta para su Master-Detail

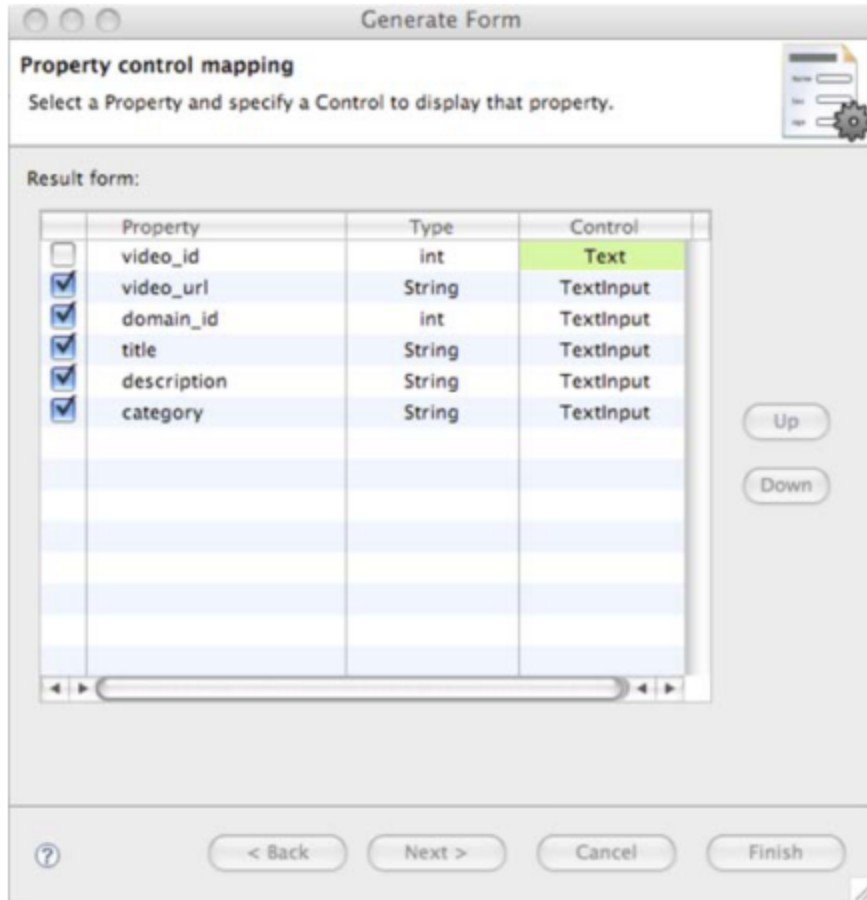


Figura 15.18 Asignaciones de control de la propiedad desde el asistente de Generación de Master-Form Detail

15.6.3 Revisión de Código de Flash Builder

Una cosa tuvo que ser cambiada manualmente, el cambio de función de gestión para la red de datos necesita establecer el valor de ID de vídeo manualmente desde IDobjeto a `dataGrid.selectedItem.video_id`. Se puede notar algo más, que falta también. No hay un control o una función para actualizar el elemento seleccionado! Usted todavía debe ser capaz de ejecutar la aplicación en este punto, pero los valores cambiantes en el formulario principal-detalle no tendrán ningún efecto en el registro de base de datos.

19 - Patrones de diseño Arquitectónico

19,1 Patrones de diseño impulsadas en Flex

Una de las cosas que se puede apreciar en la construcción de las aplicaciones Flex, son las arquitecturas más eficaces y patrones de diseño adaptadas a las necesidades y usabilidad de flex, basándose en aquellos patrones que utilizan diseño orientado a eventos y vinculan objetos, tal como es Patrón Modelo-Vista-Controlador (MVC), así como el patrón de diseño modelo de presentación son especialmente útiles para trabajar con aplicaciones Flex a gran escala.

19.1.1 El patrón Modelo-Vista-Controlador

Más específicamente, el código está organizado en tres capas: los datos encapsulados (modelo), de cara al usuario la lógica de visualización (vista), y una tercera capa que actúa como una especie de mediador entre las partes (controlador).

VISUALIZAR EL MODELO-Vista-Controlador

En términos generales, la vista a menudo tiene la capacidad de afectar directamente el modelo, pero el modelo debe pasar por el controlador con el fin de tener un efecto sobre la vista. Por esta razón el patrón MVC usualmente se muestra en la forma de un triángulo con las flechas, como se ha demostrado en la figura 19.1.

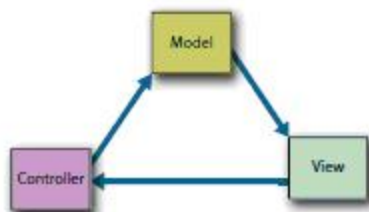


Figure 19.1 MVC is usually diagrammed in the form of a triangle.

Las flechas mostradas en el diagrama de la figura 19.1 implican que el flujo de control se mueve de modelo al controlador, entonces el controlador para ver, y luego ver de nuevo a modelo.

La figura 19.2 proporciona una representación más exacta de lo que se ve en la práctica.

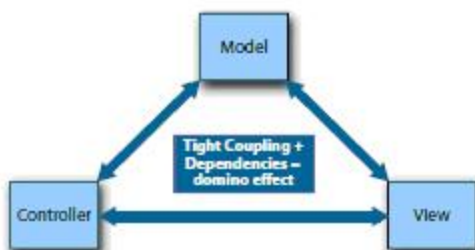


Figure 19.2 Chaotic implementations of MVC architecture lead to a domino effect of regressions caused by minor code changes.

El propósito del esquema de la figura 19.2 es señalar que las implementaciones caóticas del MVC que no tienen definido claramente las convenciones arquitectónicas en todo el proyecto causan dependencias entre elementos no relacionados, lo que resulta en un efecto dominó de regresiones cada vez que hay un cambio de código menores. Es imposible añadir o modificar la funcionalidad de la aplicación sin introducir errores

en los lugares más inesperados de la aplicación.

19.1.2 Cómo crear su propia arquitectura

A medida que la aplicación crece, también lo será su nivel de complejidad, lo que permite un conocimiento decisión a tomar con respecto a la arquitectura y / o aplicación.

El diagrama de la arquitectura que se muestra en la figura 19.3 proporciona un ejemplo de una improvisada Aplicación MVC para una aplicación prototipo que suele ser reprogramado después de la primera iteración de la microarquitectura como Robotlegs.

Para entender cómo funciona esta aplicación, se debe entender cómo las piezas se conectan entre sí.

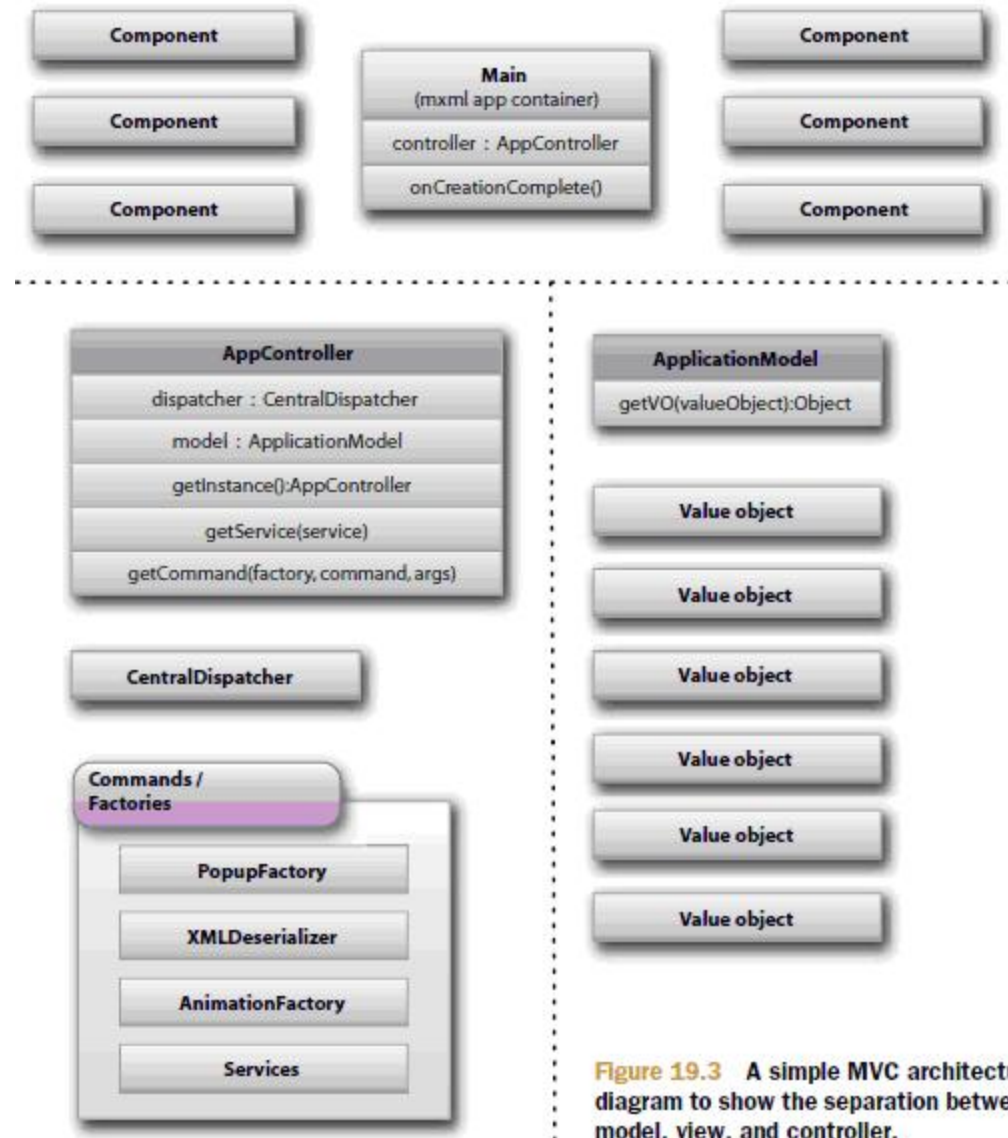


Figure 19.3 A simple MVC architectural diagram to show the separation between the model, view, and controller.

MARCO FUNDAMENTAL

La regla de oro con respecto a las mejores prácticas con los patrones de diseño es que las clases simples, los métodos estáticos y constantes no debe utilizarse si afectan el estado de aplicación.

19.2 Introducción a microarquitecturas

19.2.1 ¿Qué es una microarquitectura?

Hay una diferencia significativa entre un framework y una microarquitectura.

Flex es un marco de aplicación basado en ActionScript 3.0, mientras que Cairngorm, por ejemplo, es una microarquitectura de la estructura de Flex. La razón de mencionar esta primera es que una microarquitectura es más a menudo un marco también. El término microarquitectura refiere a una arquitectura dentro de una arquitectura mucho más grande. Uno podría razonablemente decir que múltiples arquitecturas conforman el marco global. La cosa importante a reconocer es que proporciona un cierto nivel de flexibilidad en términos de cómo Flex se utiliza para el desarrollo de aplicaciones. A medida que el equipo de Flex SDK señala, su objetivo es hacer cumplir sólo un pequeño número de las convenciones generales y no limitarse a los desarrolladores de estrictas políticas arquitectónicas. Esto es por qué han permanecido, en su mayor parte, imparcial en términos de microarquitecturas particulares. Las Gerentes de producto Flex afirman que el objetivo es apoyar el desarrollo de todas y cada una de las futuras microarquitecturas para Flex y para asegurarse de que Flex framework atiende el desarrollo progresivo y sin riesgo.

En resumen, una microarquitectura es un marco que da el paso adicional de establecer un conjunto más específico de las directrices del diseño del modelo y las políticas de arquitectura para una Aplicación Flex.

19.2.2 ¿Por qué utilizar una microarquitectura?

El uso de un microarquitectura tiene muchos beneficios, y también se introduce una cierta cantidad de riesgo.

Un análisis de costo-beneficio simple se ilustra en la tabla 19.1 para ayudarle a ver los pros y los contras de la utilización de una microarquitectura.

Table 19.1 Costs and benefits of using a microarchitecture for Flex development

Costs	Benefits
<ul style="list-style-type: none">– Adds a layer of complexity in most cases.– Effects are disastrous when not properly implemented (for example, developers who have not learned how to use the framework work on the project).– Creates a learning curve for developers who step into the project and have never used or learned the framework.– If the framework becomes obsolete, so do the developers who know how to maintain projects that implement it.	<ul style="list-style-type: none">– Enforces a set of guidelines for consistent design patterns.– Provides guidelines for organization and naming conventions.– Provides application scalability.– Code is easier to maintain.– Code can be maintained by different programmers over time.– Usually provides advantages such as decreased dependencies, loose coupling between components, enhanced performance, and code reusability.

Como se puede ver en el análisis de costo-beneficio en la tabla 19.1, utilizando una microarquitectura por lo general es más beneficioso en el desarrollo de grandes aplicaciones, que es el foco del resto del capítulo.

Cairngorm: Nos referimos a estos como microarquitecturas de primera generación, ya que fueron los primeros intentos de crear estándares de arquitectura para aplicaciones Flex empresariales.

19.2.4 microarquitecturas de segunda generación

Una de las razones principales por las que tenemos microarquitecturas y patrones de diseño en el primer lugar es el de maximizar la simplicidad en el código de aplicación.

Es por eso que es importante entender la estructura que está trabajando antes de comenzar el desarrollo de un proyecto de producción que lo utiliza.

Una característica común entre microarquitecturas de segunda generación es que son construidas para

capacitar a los desarrolladores en lugar de dominar. Actualmente las microarquitecturas más amigables de la segunda generación son Mate (francés, pronunciado: Mah-tey), Swiz framework, y uno que ha aumentado exponencialmente su popularidad, Robotlegs.

Algo que estas microarquitecturas de segunda generación tienen en común es que utilizan algún tipo de inversión de control (COI) y la inyección de dependencia(DI), definido por Martin Fowler en su libro Patrones de Arquitectura de Aplicaciones Empresariales (Addison Wesley, 2002).

19.2.5 Inversión de control e inyección de dependencia

Flex framework se basa en gran medida en los principios de inversión de control.. Esto es cierto para la mayoría de los kits de herramientas de desarrollo de interfaces de usuario basadas en eventos y marcos como Java Spring, EJB 3, Spring.NET, ColdSpring framework for ColdFusion, Needle for Ruby, y FLOW3 para PHP 5.

Con otras implementaciones como Java Spring, objetos e implementaciones de una aplicación se pueden declarar en un documento XML. Esto elimina las dependencias y los aumentos de acoplamiento debido a que la aplicación se crea dinámicamente basándose en el contenido del documento XML. Pero este método en particular no es posible con la flexión debido a la orden de las operaciones que ocurren en el inicio de la aplicación. Más específicamente, una aplicación no está listo para analizar un documento XML hasta después de sus objetos ya han sido instanciados.

Robotlegs, Swiz y Perejil todos manejan esto con metadatos en su lugar, mientras que compañero utiliza archivos de asignación MXML para contar el marco de qué hacer en el inicio.

19.3 Utilizando el marco Robotlegs

Robotlegs atiende específicamente a nivel de la empresa Flex y AS3 puro desarrollo, proporcionando una arquitectura básica sólida. Un marco como Robotlegs ofrece esta fundación a través del uso de patrones de diseño de la empresa probadas, incluyendo las siguientes:

- Optimización dependencias para promover la articulación flexible a través de "componente cableado"
- inversión del flujo de control típico de diseño orientado a objetos a través context-managed dependency injection
- mediación dinámico de los componentes para reducir la cantidad de código repetitivo que debe ser escrito bus de eventos contexto a nivel centralizado para una comunicación eficiente entre la aplicación niveles

19.3.1 Inyección de dependencias con Robotlegs

El uso estándar de la inyección de dependencia con Robotlegs proporcionará a objetos de la aplicación sus dependencias de forma rápida y sin dolor.

El suministro de un objeto con sus dependencias se logra marcando una propiedad con el [Injectar] metadato; el inyector proporcionará dicha propiedad con el valor que fue mapeado por el inyector. Este principio también se aplica a setters e constructor injection, donde tendría argumentos cuyos valores se asignan siempre a través de inyección.

Propiedad inyección:

[Inject]

```
public var myDependency:Dependency; //unnamed injection
```

La declaración de una inyección simple. Se ha mapeado una clase de tipo de Dependency, y el inyector va a inyectar ese valor en la propiedad myDependency basandose en el reglas que ha definido al configurar la asignación.

```
[Inject(name="myNamedDependency")]
public var myNamedDependency:NamedDepedency; //named injection
```

ROBOTLEGS DEPENDENCIA DE ADAPTADORES DE INYECCIÓN

Robotlegs no lleva a cabo la inyección de dependencia de forma nativa. El marco está equipado con adaptadores para y por el uso de inyección de dependencia SwiftSuspenders. La herramienta. SwiftSuspenders es una solución de rendimiento ajustado por escrito específicamente para uso con Robotlegs. A través de la utilización del modelo de adaptador pluggable, Robotlegs pueden utilizar otras soluciones DI como SmartyPants-COI o Spring ActionScript. Cuando estás solo empezando, es muy recomendable utilizar SwiftSuspenders a menos que tenga una específica necesidad de otra DI toolkit.

SwiftSuspenders hace uso de dos etiquetas de metadatos estándar para proporcionar a su objetos con puntos de inyección. Estos son [Injectar] y [PostConstruct]. El uso de estas etiquetas que son capaces de marcar las propiedades y métodos en sus clases, y Swift-Suspenders reconocerán estos y realizarán el trabajo de ellos para nosotros. La etiqueta [Injectar] es la más utilizada porque especifica los puntos de inyección, y [PostConstruct] se utiliza para marcar los métodos de ejecución después de que la clase ha sido construida y todas sus inyecciones han sido satisfechas.

19.3.2 Configuración de la inyección de dependencia con las utilidades de mapeo Robotlegs

Robotlegs tiene cuatro utilidades de mapeo base para la configuración de la inyección de dependencia en su aplicación. Cada una de estas utilidades ofrece métodos convenientes para suministrar dependencias a su aplicación. Estos mapas están disponibles en todo Robotlegs y se puede acceder normalmente a través de las clases de Context y Command.

Table 19.2 The Robotlegs MVCS classes

Name	Description
Injector	The injector is the direct adaptor to SwiftSuspenders (or the DI provider you have chosen) and implements the <code>IInjector</code> interface.
MediatorMap	The <code>MediatorMap</code> is used for defining the relationship between view components and their mediators.
CommandMap	The <code>CommandMap</code> is used for configuring commands that are triggered by events dispatched through the context.
ViewMap	The <code>ViewMap</code> is used to define view component classes that will be injected with dependencies when they're added to the stage.

INJECTOR

El inyector es un adaptador directamente a la solución DI la aplicación está utilizando. A pesar de estos ejemplos se utiliza el adaptador SwiftSuspenders de forma predeterminada, cualquier adaptador que implementa la interfaz `IInjector` puede utilizarlo. El `IInjector` interfaz proporciona cuatro métodos para configurar las inyecciones de dependencia:

- `mapSingleton` se utiliza para asignar una única instancia de una clase.
- `mapSingletonOf` se utiliza para asignar una única instancia de una clase base o interfaz.
- `mapValue` se utiliza para asignar una instancia específica de una clase.
- `mapClass` se utiliza para asignar muchas instancias únicas de la clase.

El método `mapSingleton` toma una clase como argumento y lo asigna una instancia de esta clase para

todos los puntos de inyección definidos:

```
injector.mapSingleton(MyClass);
```

El `mapSingletonOf` es similar al `mapSingleton`, pero permite definir un tipo base de la interfaz:

```
injector.mapSingletonOf(IMyClass, MyClass); //MyClass implements IMyClass
```

El método `mapValue` mapea la instancia de un simple objeto de una clase.

```
Var myClassInstance:MyClass = new MyClass();
```

```
Injector.mapValue(MyClass, myClassInstance);
```

El método `mapClass` provee una nueva instancia de la clase mapeada para cada punto de inyección de la clase. No es necesario instanciar la clase, ya que al momento de ser llamada se crea e inyecta sobre la clase requerida.

```
Injector.mapClass(MyClass);
```

Esto puede proporcionar a la aplicación una gran flexibilidad, que le permite hacer sustituciones y sacar el máximo provecho de polimorfismo. Este enfoque es muy útil cuando se quiere proporcionar sus clases inyectados con una versión de código auxiliar de una clase durante la prueba y desarrollo y rápidamente reemplazan que con otra versión de la clase en la producción.

MEDIATORMAP

La clase `MediatorMap` se utiliza para definir la relación entre los componentes de la vista y los mediadores que los conectan con el resto de la aplicación. El mediador tiene por defecto puntos de inyección los cuales serán satisfechos a través de la inyección de dependencias. Debido a su fin previsto, el `MediatorMap` tiene un único método para definir esta relación: `mapView`.

```
mediatorMap.mapView(MyAwesomeWidget, MyAwesomeWidgetMediator);
```

COMMANDMAP

Al igual que el `MediatorMap`, `CommandMap` tiene un propósito específico. Vamos a usar el `CommandMap` para definir la relación entre los eventos de `ActionScript` y `Command` classes. El `CommandMap` tiene un único método que se utiliza para definir esta relación:

```
commandMap.mapEvent(MyAppDataEvent.DATA_WAS_RECEIVED, MyCoolCommand)
```

VIEWMAP

El `viewmap` permite mapear componentes de vista para una inyección. Muy similar a `mediatorMap`, `viewmap` escucha por los componentes vista para luego añadirlos a la etapa. Diferente a `MediatorMap`, `viewmap` inyectará todas las dependencias directamente al componente vista después que sea incorporado a la etapa. Para hacer uso de él, se utiliza el siguiente llamado:

```
viewMap.mapType(MyViewComponent);
```

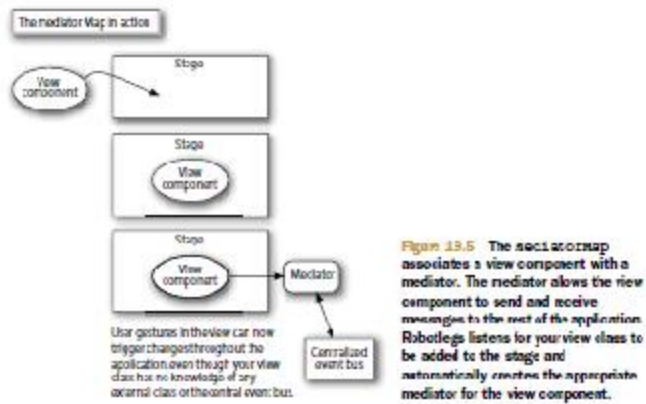


Figure 19.5 The mediator map associates a view component with a mediator. The mediator allows the view component to send and receive messages to the rest of the application. Robotlegs listens for your view classes to be added to the stage and automatically creates the appropriate mediator for the view component.

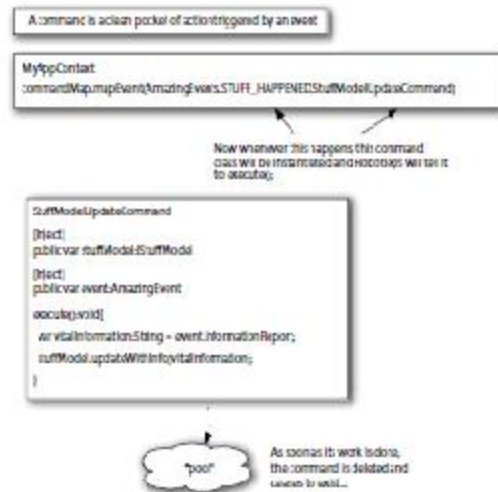


Figure 19.6 Mapped commands are executed by the CommandMap when the mapped event is dispatched through the central event bus. Robotlegs provides the command with all of its injections prior to running the execute() method. Robotlegs temporarily maps the event that triggered the command for injection to provide access to any custom event properties/ payloads you have provided.

19.4 Creación de aplicaciones con MVCS Robotlegs

Crear su primera aplicación a nivel empresarial con la implementación Robotlegs MVCS.

Es un enfoque lightly prescriptive que posibilita la construcción de una aplicación que tendrá una estructura arquitectónica sólida.

En una aplicación MVCS el modelo almacena la información, la vista proporciona un mecanismo para que el usuario pueda interactuar con el controlador y este responda las acciones, y el servicio se conecta a los recursos externos, tales como un servicio web o un archivo local de sistema.

MVCS Robotlegs fue inspirado por la estructura de PureMVC con la adición del nivel de servicio. Dentro de la implementación MVCS son cuatro clases principales, como se ilustran a continuación.

Contexto	Mediador	Comando	Actor
El contexto de clase es un hub. Facilita la comunicación entre niveles y proporciona la secuencia de arranque inicial de la estructura.	Mediador sirven como conectores entre los componentes de vista. Sus usuarios interactúan con y el marco, y representan el punto de vista más alto.	Sirven como un mecanismo para reducir los niveles de su aplicación y encapsular la lógica común. Representan el nivel de controlador.	Actor proporcionan parte de la funcionalidad básica que se utiliza por las clases, tanto en el servicio y los niveles de modelo.

Cada una de estas clases cumple un rol importante en la aplicación. Es una forma eficiente de trabajar ya que posibilita la separación de las diversas responsabilidades en estos niveles.

No hay clases de modelo o Servicio. Esto se debe a Robotlegs no proporciona implementaciones de estas clases. Estas son utilizadas para separar las responsabilidades y ambos utilizan la clase Actor.

19.4.1 Configuración de un proyecto Robotlegs

La única diferencia importante con un proyecto Robotlegs es que usted tendrá que añadir la biblioteca a la carpeta lib de tu proyecto. Para hacerlo descargue la última biblioteca Robotlegs. Se puede encontrar en <http://www.robotlegs.org>. Con la biblioteca descargada, crear un nuevo proyecto en Flash Builder llamada Robotlegs-ContactManager con el tipo de aplicaciones web.

- Arrastre el archivo SWC Robotlegs en la carpeta libs del proyecto nuevo.
- Su nuevo proyecto ahora tiene acceso a todas las clases Robotlegs.

Name	Phone Number
Tariq Ahmed	555-555-1111
John Bland	555-555-1212
Dan Orlando	555-555-1313
Joel Hooks	555-555-1414

Edit Contact Name: Phone Number: Save Contact Cancel

Figura 19.7 es el gestor de contactos simple que estamos construyendo aquí.

Ahora que el proyecto está configurado, está listo para comenzar a añadir clases al contact manager, comenzando con la clase Context.

19.4.2 Secuencia de arranque de la aplicación con la clase Context

La clase **Context**, como se muestra en la tabla 19.3, se puede considerar *el corazón* de la aplicación Robotlegs. Es el punto de entrada para el framework y crea instancias de los servicios descritos en el apartado anterior. No está limitado a un solo contexto dentro de una aplicación, pero para muchos casos de uso, un contexto será suficiente. La capacidad de usar múltiples contextos es esencial para las aplicaciones modulares. Vamos a centrarnos en el uso de un único contexto en una aplicación no modular.

```
package com.fx4ia.robotlegs.contacts
```

```
{
```

```
    import com.fx4ia.robotlegs.contacts.controller.*;
```

```
    import com.fx4ia.robotlegs.contacts.events.ContactEvent;
```

```
    import com.fx4ia.robotlegs.contacts.model.ContactsModel;
```

```
    import com.fx4ia.robotlegs.contacts.service.*;
```

```
    import com.fx4ia.robotlegs.contacts.service.events.ContactServiceEvent;
```

```
import com.fx4ia.robotlegs.contacts.view.*;
```

```
import org.robotlegs.mvcs.Context;
```

```
public class ContactManagerContext extends Context
```

```
{
```

```
    override public function startup():void
```

```
    {
```

```
        injector.mapSingleton(ContactsModel);
```

Mapeo IContactService

```
        injector.mapSingletonOf(IContactService,
```

```
        XMLContactService);
```

Mapeo las vistas

```
        mediatorMap.mapView(ContactsView,
```

```
        ContactsViewMediator);
```

```
        mediatorMap.mapView(EditContactView,
```

```
        EditContactViewMediator);
```

```
        mediatorMap.mapView(ContactsToolbarView,
```

```
        ContactsToolbarViewMediator);
```

```
        commandMap.mapEvent(ContactServiceEvent.LOAD,
```

Mapeo los comandos

```
LoadContactsCommand, ContactServiceEvent);
```

```
commandMap.mapEvent(ContactEvent.UPDATE,
```

```
SaveContactCommand, ContactEvent);
```

```
commandMap.mapEvent(ContactEvent.CREATE,
```

```
CreateContactCommand);
```

```
commandMap.mapEvent(ContactEvent.CANCEL_CREATE,
```

```
CreateContactCancelCommand);
```

```
commandMap.mapEvent(ContactEvent.DELETE,
```

```
DeleteContactCommand);
```

```
commandMap.mapEvent(ContactEvent.EDIT,
```

```
EditContactCommand);
```

```
commandMap.mapEvent(ContactEvent.CANCEL_EDIT,
```

```
EditContactCancelCommand);
```

```
commandMap.mapEvent(ContactEvent.SELECT,
```

```
SelectContactCommand);
```

```
}
```

```
}
```

```
}
```

Listado 19.3 muestra el ContactManagerContext creado en el paquete com.fx4ia.robotlegs.contacts que servirá como el paquete raíz para nuestro ejemplo.

Al crear el Contexto de la aplicación, reemplazaremos el método startup(). Este método es llamado por Robotlegs cuando el contexto se ha inicializado totalmente. En este momento usted tendrá acceso al mapeo de las dependencias. El contexto, es un lugar conveniente para arrancar la aplicación.

En ContactManagerContext estamos mapeando el **modelo**, el **servicio**, los componentes de la **vista**, y sus **mediadores**, así como los diversos comandos a sus respectivos eventos que los provocan. En aplicaciones de mayor tamaño este boot-startup se coloca a menudo en los **comandos** que permitan separar el método de inicio del contexto.

Para este ejemplo, la colocación de estos mapeos aquí será suficiente. Hablaremos de las clases que se están asignando y que hacen.

Su contexto se inicializa dentro del componente de vista. En general, este es el componente raíz.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
```

```
xmlns:s="library://ns.adobe.com/flex/spark"
```

```
xmlns:mx="library://ns.adobe.com/flex/mx"
```

```
xmlns:contacts="com.fx4ia.robotlegs.contacts.*"
```

```
xmlns:view="com.fx4ia.robotlegs.contacts.view.*">
```

1.

Se crea el ContactManagerContext declarada en esta etiqueta

```
<fx:Declarations>
```

```
<contacts:ContactManagerContext
```

```
contextView="{this}"/>
```

```
</fx:Declarations>
```

(2) Se define el Contexto de la vista

```
<s:layout>
```

```
<s:VerticalLayout/>
```

```
</s:layout>
```

Las vistas

```
<view:ContactsToolbarView/>
```

```
<view:ContactsView/>
```

```
<view:EditContactView/>
```

```
</s:Application>
```

Listado 19.4 muestra un contexto Robotlegs inicializado mediante una etiqueta MXML.

Al igual que con todos los elementos no visuales en MXML, a su contexto hay que añadir dentro de las declaraciones el tag MXML (1). También se puede crear la instancia de contexto con Acción-Script con un evento con un ciclo de vida, pero es conveniente utilizar MXML para crearla. El contexto requiere una referencia de una vista, que la provee la propiedad ContextView (2) y no se iniciara automáticamente hasta que uno la indique. Este componente de vista actúa como ámbito de la aplicación, y muchas cosas se producen en este ámbito. No hay ningún script, y todos los elementos visuales están contenidos en clases. La aplicación es un punto de partida y nada más. El trabajo que el usuario va a hacer se produce en los componentes y las clases bien encapsuladas en otros lugares. Eso es todo. Se está utilizando Robotlegs. No estamos haciendo mucho, pero hemos creado un contexto sobre el cual construir el resto de la aplicación.

Para el siguiente paso, vamos a mirar los componentes de vista y cómo trabajar con ellos, para que puedan comunicarse con otros mediadores y niveles de aplicación.

19.4.3 Mediación de vistas

Los **mediadores** son el nombre del patrón de diseño que ellos representan.

Se podría considerar un mediador, como la oficina de correos.

Por ejemplo, su madre ha escrito una linda carta.

Ella lo deja en el buzón de correo y el cartero lo recoge.

Viaja por el país hasta que se le es entregado.

Usted no quiere caminar al otro lado del país, hasta la casa de su madre para obtenerla y ella ciertamente no tiene la energía para entregársela en mano.

De este modo, la oficina de correos ha mediado en el proceso de entrega de la carta.

A este **patrón** de diseño se le llama **Mediator**, funciona de forma similar mediante la entrega y recepción de mensajes entre las clases de la aplicación. Con la implementación Robotlegs MVCS, los mediadores representan un nivel más que las vistas. Cada componente de vista puede llegar a tener un mediador que le sirve de puerta de entrada a otros componentes de la vista y para los otros niveles dentro de la aplicación. Este tipo de control granular, con un mediador para cada componente de vista, es probable que no sea necesario. En su lugar, se mediará los componentes principales. Tomemos, por ejemplo, un formulario. Usted no mediará cada TextInput, DropDownList, y Button que componen el formulario. En su lugar, usted mediará el formulario como un componente completo. Echemos un vistazo a la ContactsView (ver figura 19.7 en la vista Contactos se marca como 2), que contiene una cuadrícula de datos para mostrar la lista de contactos en el próximo listado.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<s:Group xmlns:fx="http://ns.adobe.com/mxml/2009"
```

```
xmlns:s="library://ns.adobe.com/flex/spark"
```

```
xmlns:mx="library://ns.adobe.com/flex/mx"
```

```
width="100%" height="100%" enabled="{_viewEnabled}">
```

```
<fx:Script>
```

```
<![CDATA[
```

```
import com.fx4ia.robotlegs.contacts.events.ContactEvent;
```

```
import com.fx4ia.robotlegs.contacts.model.vo.Contact;
```

```
import mx.collections.IList;
```

```
import mx.events.ListEvent;
```

```
[Bindable]
```

```
public var dataProvider:IList;
```

```
[Bindable]
```

```
private var _viewEnabled:Boolean = true;
```

```
public function set viewEnabled(value:Boolean):void
```

```
{
```

```
    _viewEnabled = value;
```

```
    list.selectedItem = null;
```

```
}
```

```
public function selectContact(contact:Contact):void
```

```
{
```

```
    list.selectedItem = contact;
```

```
}
```

```
protected function list_changeHandler(event>ListEvent):void
```

```
{
```

```
dispatchEvent(new ContactEvent(ContactEvent.SELECT, selected));
```

```
}
```

```
protected function list_doubleClickHandler(event:MouseEvent):void
```

```
{
```

```
dispatchEvent(new ContactEvent(ContactEvent.EDIT, selected));
```

```
}
```

```
protected function get selected():Contact
```

```
{
```

```
return list.selectedItem as Contact;
```

```
}
```

```
]]>
```

```
</fx:Script>
```

```
<mx:DataGrid id="list" width="100%" height="100%"
```

```
doubleClickEnabled="true"
```

```
dataProvider="{dataProvider}"
```

```
change="list_changeHandler(event)"
```

```

doubleClick="list_doubleClickHandler(event)">

<mx:columns>

<mx:DataGridColumn id="nameColumn" dataField="name"

headerText="Name"/>

<mx:DataGridColumn id="phoneNumberColumn" dataField="phoneNumber"

headerText="Phone Number"/>

</mx:columns>

</mx:DataGrid>

</s:Group>

```

El **ContactsView** es un componente típico. Es un grupo que se ajusta un DataGrid. La propiedad dataProvider es enlazable, y el propio DataGrid enlaza su propiedad dataProvider a este valor. Cuando se selecciona un elemento, el controlador envía un *evento* ContactEvent.SELECT con el contacto seleccionado. Del mismo modo, como una conveniencia para nuestros usuarios, cuando un elemento se hace doble clic, un ContactEvent.EDIT es enviado. Como veremos en breve, el mediador de este componente está a la escucha para estos eventos. Veamos ahora el Mediador y averiguar qué tipo de trabajo que está haciendo por nosotros en la siguiente lista.

```

package com.fx4ia.robotlegs.contacts.view

```

```

{

```

```

import com.fx4ia.robotlegs.contacts.events.ContactEvent;

```

```
import com.fx4ia.robotlegs.contacts.model.ContactsModel;
```

```
import com.fx4ia.robotlegs.contacts.model.events.ContactsModelEvent;
```

```
import com.fx4ia.robotlegs.contacts.service.events.ContactServiceEvent;
```

```
import flash.events.Event;
```

```
import org.robotlegs.mvcs.Mediator;
```

```
public class ContactsViewMediator extends Mediator
```

```
{
```

```
    [Inject]
```

```
    public var view:ContactsView;
```

```
    [Inject]
```

```
    public var model:ContactsModel;
```

```
    override public function onRegister():void
```

```
    {
```

```
        eventMap.mapListener( eventDispatcher, ContactServiceEvent.LOADED,
```

```
        handleContactsLoaded );
```

```
        eventMap.mapListener( eventDispatcher, ContactServiceEvent.SAVED,
```

```
handleContactSaved)
```

```
eventMap.mapListener( eventDispatcher, ContactEvent.CREATE,
```

```
disableViewOnEvent);
```

```
eventMap.mapListener( eventDispatcher, ContactEvent.CANCEL_CREATE,
```

```
enableViewOnEvent);
```

```
eventMap.mapListener( view, ContactEvent.SELECT,
```

```
dispatch );
```

```
eventMap.mapListener( view, ContactEvent.EDIT,
```

```
dispatch );
```

```
dispatch(new
```

```
ContactServiceEvent(ContactServiceEvent.LOAD));
```

```
}
```

```
protected function handleContactsLoaded(event:ContactServiceEvent):void
```

```
{
```

```
view.dataProvider = model.list;
```

```
}
```

```
protected function handleContactSaved(event:ContactServiceEvent):void

{

    enableViewOnEvent(event);

    view.selectContact(model.editing);

}

protected function disableViewOnEvent(event:Event):void

{

    view.viewEnabled = false;

}

protected function enableViewOnEvent(event:Event):void

{

    view.viewEnabled = true;

}

}
```

Un Mediator tiene una relación uno-a-uno con su respectiva vista. Por esta razón es importante asociar el

componente de vista en el mediador. No se preocupe por la asignación de esta relación. El **MediatorMap** crea la relación entre el componente de vista y su mediador en el contexto. Además de la relación del componente de vista, te darás cuenta de que se está relacionando la ContactsModel. Como discutiremos en la sección 19.5.5, el modelo controla el estado de la aplicación y se accede por el mediador. El método **onRegister** se reemplazara en casi todos los casos. Este método es el gancho en el mediador. Cuando se llama a este método, usted puede estar seguro de que las relaciones se han incluido y que el mediador está totalmente inicializado. Dentro del método onRegister utiliza el **EventMap** para añadir escuchas tanto al marco de la aplicación como a la vista que el mediador quiera conectar.

Escucha los eventos y trata de que la vista y el marco de aplicación en general no sepan uno del otro. Sirve como un puente y proporciona una separación limpia de sus responsabilidades. Desde el punto de vista del usuario no es necesario preocuparse por la manipulación de datos y lógica ya que ocurre dentro de la aplicación. Promueve el diseño de componentes reutilizables. Esta es la primera vez que ve EventMap.

El **EventMap** es un objeto local creado por MVCS Mediador y clases Actor. Es una clase de utilidad que se utiliza para eventos del mapa. Su objetivo principal es proporcionar un mecanismo para crear y eliminar automáticamente los detectores de eventos. Al utilizar el EventMap no es necesario llamar a removeEventListener para cada oyente que se haya registrado. Esto sirve para el doble propósito de reducir código repetitivo, así que ayuda a reducir las pérdidas de memoria que pueden ocurrir al eliminar incorrectamente los detectores de eventos. El EventMap proporciona el método unmapListeners para eliminar a la vez todos los detectores registrados. Un mediador invoca esta función automáticamente cuando es eliminado.

Usar EventMap para garbage collection amigable con event listening

EventMap es muy cómodo a la hora de asignar detectores de eventos dentro de las clases Robotlegs MVCS. Proporciona una sintaxis concisa y también incluye el método unmapListeners para eliminar todos los detectores de eventos. Dejando a los detectores de eventos sin la eliminación adecuada, puede dar lugar a algunas pérdidas de memoria graves. Los listeners a los que no se les eliminan las referencias de las asociaciones a las clases, no son tenidos en cuenta a la hora de eliminarlos del garbage collection de Flash Player.

El EventMap escucha eventos de la propiedad de eventDispatcher del Mediador. El eventDispatcher se encuentra dentro de todas las clases MVCS y es un bus de eventos centralizado que es utilizado por Robotlegs para la comunicación entre las clases dentro de la aplicación. Cada contexto proporciona un único IEventDispatcher para facilitar esta comunicación. No es algo necesario para crear o proporcionar, el contexto lo hace el mismo. La figura 19.8 ilustra cómo el bus evento central facilita la comunicación entre los objetos dentro de la aplicación. En el componente ContactsView enviamos dos eventos, ContactEvent.SELECT y ContactEvent.EDIT. Ambos eventos han sido asignados en el mediador a través del EventMap.

Ya sabemos que el **handler** es un método llamado por despacho. Este es otro método conveniente proporcionado por clases MVCS para simplificar la sintaxis necesaria para distribuir un evento a través del IEventDispatcher del Contexto. En lugar de escribir la larga EventDispatcher.dispatchEvent(event), se puede escribir dispatch(event). Cuando se utiliza como un handler de eventos, funciona como una especie de relay para distribuir el evento recibido de la vista directamente al evento distribuidor del framework.

Tenga en cuenta que la vista no tiene una conexión directa al bus evento central. Se combina con una o MEDIAT,

que se encarga de la comunicación con la aplicación en su conjunto.

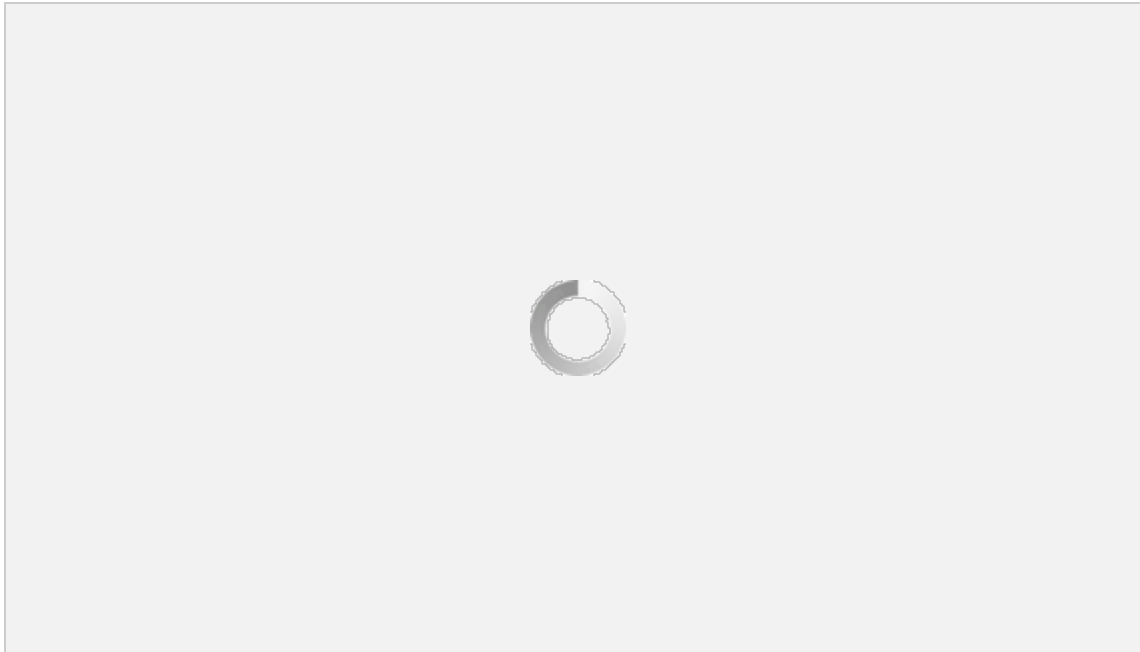


Figura 19.8 En lugar de escuchar eventos directamente el uno del otro, los objetos utilizan el bus del evento central para comunicarse. Actores, Mediadores y comandos están equipadas con acceso a este bus de eventos para permitir la comunicación desacoplada.

Nombre	Descripcion
<i>CreateContactCommand</i>	Crea un nuevo Contact vacío para ser editado por el usuario.
<i>CreateContactCancelCommand</i>	Cancela la creación de un nuevo Contact.
<i>DeleteContactCommand</i>	Borra un Contact
<i>EditContactCommand</i>	Inicializa un Contact para ser editado.
<i>EditContactCancelCommand</i>	Cancela la edición de un Contact.
<i>LoadContactsCommand</i>	Carga una colección de Contacts desde un service.
<i>SaveContactCommand</i>	Guarda un Contact por el servicio.

<i>SelectContactCommand</i>	Inicializa un Contact.
-----------------------------	------------------------

Tabla 19.4 Comandos que componen el controlador de nivel en el ejemplo Administrador de Contactos.

```
package com.fx4ia.robotlegs.contacts.controller
```

```
{
```

```
    import com.fx4ia.robotlegs.contacts.events.ContactEvent;
```

```
    import com.fx4ia.robotlegs.contacts.model.ContactsModel;
```

```
    import org.robotlegs.mvcs.Command;
```

```
    public class SelectContactCommand extends Command
```

```
    {
```

```
        [Inject]
```

Relacionamos el evento

```
        public var event:ContactEvent;
```

```
        [Inject]
```

Relacionamos el Modelo

```
        public var model:ContactsModel;
```

```
        override public function execute():void
```

```
        {
```

Damos el evento al modelo para que lo trate

```
        model.selected = event.contact;
    }

}

}
```

Los comandos de este ejemplo son todos relativamente cortos. No representan una gran cantidad de funcionalidades, y la mayoría sólo son una línea de código. El ejemplo es relativamente simple, sin embargo, los comandos pueden contener y hacer más responsabilidades. Es una buena meta tratar de mantener que las responsabilidades de un comando sean las mínimas posibles. En el caso de la `SelectContactCommand` esperamos que haga una cosa y eso es seleccionar un contacto. Al distribuir un evento a través del `event dispatcher` del Contexto, se asigna a un comando, es manejado por el `CommandMap`. El **`CommandMap`** responderá este evento mediante la creación de una instancia de la clase `Command` asignado a ese tipo de evento, dotándolo de las propiedades del mapeo, lo que incluye la instancia del evento que desencadenó el comando, y finalmente, se ejecutará el método `execute()` del comando. Consulte la figura 19.6, que ilustra esta relación. Esto no es estrictamente necesario para utilizar la *clase* `Command` MVCS. El único requisito de un comando es que tiene un método `execute()`. El *Comando* MVCS proporciona otras propiedades que incluyen el mapeo de utilidades de `Robotlegs`, asignar servicios, un método `dispatch()`, y el acceso a la `ContextView` del contexto. Es importante tener en cuenta que los comandos están destinados a ser de corta duración. Son creados para hacer su trabajo, y luego se destruyen rápidamente. Los comandos son útiles para separar la vista de nivel, los mediadores y sus componentes, a partir de los modelos y niveles de servicio. Tanto el modelo y las clases de servicio pueden ser inyectados y acceder a los mediadores, pero esta práctica no es nada recomendable, ya que no es el trabajo de los mediadores para establecer los datos, mantener el estado de la aplicación, o hacer peticiones a distancia a través de un servicio. Al hacer uso de los comandos, la aplicación va a ser infinitamente más escalable y fácil de administrar en el largo plazo. Hemos echado un vistazo a las clases de mediadores y Comando. Ahora echemos un vistazo a los dos niveles restantes de una aplicación MVCS, empezando por los servicios.

19.4.5 Los servicios son la puerta de entrada al mundo

Un **servicio** es algo fuera de nuestra solicitud que tenemos que acceder o interactuar. El ejemplo obvio de un servicio es un servicio web que proporciona acceso remoto a una base de datos. Esto incluye aplicaciones remotas como `LiveCycle Data Services`, APIs web desde sitios web como `Google` o `Flickr`, o una serie de otras aplicaciones Web remotas que ofrecen algún tipo de método para la recuperación de datos. Para nuestro gestor de contactos, estamos usando un servicio de simulacro. Los datos se encuentran en un archivo XML que está integrado en la aplicación, y no se realizan hay llamadas remotas. Este enfoque puede ser útil en las primeras etapas del desarrollo, lo que le permite trabajar en la parte del cliente de la aplicación incluso antes de que existan servicios reales. También es un enfoque esencial cuando se trata de ofrecer pruebas unitarias con servicios remotos, es defectuosa e incómoda en el mejor de los casos. Echa un vistazo a los datos XML ubicados

en la carpeta de datos que está siendo utilizado por la aplicación:

```
<Contacts>
  <contact id="1" name="Tariq Ahmed" phoneNumber="555-555-1111"/>
  <contact id="2" name="Juan Sosa" phoneNumber="555-555-1212" />
  <contact id="3" name="Dan Orlando" phoneNumber="555-555-1313"/>
  <contact id="4" name="Joel Hooks" phoneNumber="555-555-1414"/>
</Contacts>
```

Esta información no es compleja, pero es un punto de partida y se puede ampliar fácilmente a medida que la aplicación crece. En un gestor de contactos reales es probable que se desee información más detallada, como direcciones, fotografías de sus contactos y otros detalles acerca de sus asociados.

```
package com.fx4ia.robotlegs.contacts.service
```

```
{
```

```
    import com.fx4ia.robotlegs.contacts.model.ContactsModel;
```

```
    import com.fx4ia.robotlegs.contacts.model.vo.Contact;
```

```
    import com.fx4ia.robotlegs.contacts.service.events.ContactServiceEvent;
```

```
    import com.fx4ia.robotlegs.contacts.service.helpers.ContactXMLParser;
```

```
    import mx.collections.ArrayCollection;
```

```
    import org.robotlegs.mvcs.Actor;
```

```
    public class XMLContactService extends Actor
```

Extiende al Actor e implementa IContactService

```
        implements IContactService
```

```
{
```

```
[Embed(source="/data/contacts.xml",mimeType="text/xml")]
```

```
protected var ContactsXML:Class;
```

Agrego la información e ContactsXML

```
[Inject]
```

```
public var model:ContactsModel;
```

Relaciono el modelo

```
public function load():void
```

```
{
```

```
var xml:XML = XML( ContactsXML.data );
```

```
model.list =
```

```
ContactXMLParser.getCollection(xml);
```

```
dispatch(
```

Despacho el servicio con un nuevo evento

```
new ContactServiceEvent(ContactServiceEvent.LOADED));
```

```
}
```

```
public function save(contact:Contact):void
```

```
{
```

```
if(!model.list.contains(contact))
```

```
model.list.addItem(contact);
```

```
if(contact.id == 0)
```

```
getNextId(contact);
```

```
model.editing = null;
```

```
dispatch(new ContactServiceEvent(ContactServiceEvent.SAVED));
```

```
}
```

```
protected function getNextId(forContact:Contact):void
```

```
{
```

```
var id:int = 0;
```

```
for each(var contact:Contact in model.list)
```

```
{
```

```
if(contact.id > id)
```

```
id = contact.id;
```

```
}
```

```
forContact.id = id++;
```

```

    }

    public function remove(contact:Contact):void

    {

        model.remove(contact);

    }

}

```

Listado 19.8 es la clase de servicio que carga los datos anteriores.

Cuando estudiamos el servicio dentro del método del Contexto, vemos que se utilizó el método mapSingletonOf del inyector para asignar la instancia de servicio. Observe que el XMLContactService implementa la interfaz IContactService. Esto nos permite cambiar una sola línea, la asignación en el contexto, utilizar otro IContactService. Cuando llegue el momento de conectar la aplicación a un servicio directo, puede cambiarlo en cuestión de segundos. Este enfoque es muy potente y permite mucha flexibilidad. Además de implementar IContactService, XML-ContactService extiende la clase Actor MVCS Robotlegs. **Actor** es una clase de utilidad que proporciona eventos del contexto y el método dispatch (). Al hacer uso de servicios externos es importante analizar y convertir los datos que se recuperan tan pronto como sea posible. En el método load () de este servicio se ve que se está utilizando un ContactXMLParser. Esta clase de ayuda necesita los datos XML y crea una ArrayCollection de objetos de contacto. No queremos hacer frente a los datos XML genéricos cuando podemos utilizar objetos de valor inflexible de tipos específicos de nuestra aplicación. Además de cargar los datos iniciales, este servicio ofrece la posibilidad de guardar y eliminar contactos del servicio. Si se conecta a un servicio remoto, estaría llevando a cabo estas acciones de forma asíncrona con controladores de eventos o respuestas de peticiones a una HTTPService, RemoteObject o NetConnection. Aunque este ejemplo, se establece inmediatamente los datos en el modelo, el principio es el mismo. El último nivel de MVCS es el modelo que gestiona los datos de nuestra aplicación y el estado actual de ellos. Vamos a echar un vistazo al modelo actual.

19.4.6 Utilizando el modelo para la gestión de datos y el estado

Los modelos representan los datos. Los datos representan el estado de su solicitud. Los elementos seleccionados, el contacto que comenzaron a editarse, cambios en la lista de contactos, y otros cambios que afectan el estado representados de la aplicación se producirá a través del modelo. Al igual que los servicios, los modelos no tienen una clase que los representen específicamente. La ampliación de la clase Actor Robotlegs MVCS los crea. Ambos servicios y modelos son los niveles conceptuales en Robotlegs MVCS. Esto le

proporciona una gran cantidad de flexibilidad en cuanto a la implementación de estos niveles dentro de la aplicación.

```
package com.fx4ia.robotlegs.contacts.model
```

```
{
```

```
    import com.fx4ia.robotlegs.contacts.events.ContactEvent;
```

```
    import com.fx4ia.robotlegs.contacts.model.events.ContactsModelEvent;
```

```
    import com.fx4ia.robotlegs.contacts.model.vo.Contact;
```

```
    import com.fx4ia.robotlegs.contacts.service.events.ContactServiceEvent;
```

```
    import mx.collections.ArrayCollection;
```

```
    import org.robotlegs.mvcs.Actor;
```

El Modelo extiende al Actor

```
    public class ContactsModel extends Actor
```

```
    {
```

```
        private var _list:ArrayCollection;
```

```
        public function get list():ArrayCollection
```

```
        {
```

```
            return _list;
```

```
}
```

```
public function set list(value:ArrayCollection):void
```

```
{
```

```
    _list = value;
```

```
}
```

Contacto seleccionado

```
private var _selected:Contact;
```

```
public function get selected():Contact
```

```
{
```

```
    return _selected;
```

```
}
```

```
public function set selected(value:Contact):void
```

```
{
```

```
    _selected = value;
```

```
    dispatch(new
```

Despacho el evento del contacto

```
ContactsModelEvent(ContactsModelEvent.SELECTED));
```

```
}
```

```
private var _editing:Contact;
```

```
public function get editing():Contact
```

```
{
```

```
    return _editing;
```

```
}
```

Comienzo a editar el contacto

```
public function set editing(value:Contact):void
```

```
{
```

```
    _editing = value;
```

```
    dispatch(new ContactsModelEvent(ContactsModelEvent.EDITING));
```

```
}
```

Metodo para crear un nuevo contacto

```
public function create():Contact
```

```
{
```

```
    var contact:Contact = new Contact();
```

```
    editing = contact;
```

```
dispatch(new ContactsModelEvent(ContactsModelEvent.CREATED));
```

```
return contact;
```

```
}
```

Método para borrar un contacto

```
public function remove(contact:Contact):void
```

```
{
```

```
var contactIndex:int = list.getItemIndex(contact);
```

```
if(contactIndex > -1) list.removeItemAt(contactIndex);
```

```
if(selected == contact) selected = null;
```

```
if(editing == contact) editing = null;;
```

```
dispatch(new ContactsModelEvent(ContactsModelEvent.REMOVED));
```

```
}
```

```
}
```

```
}
```

Listado 19,9 ContactsModel.

Cuando un usuario realiza una acción dentro de la aplicación, tales como la selección de un contacto en la lista, se activa un comando. El comando posteriormente accede al modelo y se define la propiedad seleccionada. El alojamiento se encuentra a través de un método setter, y cuando cambia la selección, elvContactsModel

despacha el evento `ContactsModel-Event.SELECTED` a través de eventos despachados por el framework. Este evento es luego recibido por los interesados, incluidos los mediadores, y tal vez desencadena otro comando. Además de la gestión de estos temas relacionados con el Estado, el modelo se encarga de manipulación de los datos. Cuando se crea o se elimina un contacto de la lista de contactos, se hace en el modelo. Al contener estas acciones dentro del modelo, puede encapsular los datos a manipular. No se extiende a lo largo de toda la aplicación. Esta encapsulación hace que sea mucho más fácil de depurar la aplicación y crea claridad durante el proceso de desarrollo.

Ahora que ya has visto cada una de las capas de una aplicación Robotlegs MVCS, debes tener una buena idea de cómo funciona. El código fuente completo de este ejemplo está disponible, y probablemente querrás examinarlo para ver la aplicación en ejecución. No nos fijamos en todas las clases de la aplicación para evitar la repetición, pero esta visión general cubre todos los aspectos básicos que usted necesita para empezar a escribir aplicaciones Robotlegs propias. El uso de un framework como Robotlegs le permitirá programar aplicaciones complejas con mayor rapidez y crear patrones y prácticas para que usted y su equipo.

Visita <http://robotlegs.org> para más ejemplos y enlaces a apoyar foros.