

Personalizando la experiencia.

Nos vamos a enfocar en el diseño de experiencia del usuario y cómo utiliza las técnicas de UXD/UED (User Experience Design) en Flex.

Principios de diseño de experiencia del usuario.

El diseño de experiencia del usuario, (User experience design, UXD), tiene sus raíces en factores humanos y ergonómicos que estudia el modo en que los humanos interactúan con las máquinas en el esfuerzo de construir mejores sistemas.

El término fue acuñado por Don Norman cuando era el Vice Presidente de Advanced Technology Group en Apple. Dijo : "I invented the term because I thought human interface and usability were too narrow"

UXD ha evolucionado para convertirse en un campo multidisciplinario que comprende aspectos de psicología, antropología, ciencias de la computación, diseño gráfico, diseño industrial y ciencia cognitiva.

Nos centraremos en principios específicos del UXD relacionados a la programación de software que puede aplicar a sus aplicaciones Flex.

Construyendo alrededor de las historias del usuario.

Las historias del usuario se originan - y están estrechamente relacionadas - con el concepto de escenarios de casos de uso.

Sin embargo, con los escenarios de casos de uso, un desarrollador pretende identificar el uso esperado de la aplicación del usuario. Eventualmente la gente comenzó a darse cuenta de que esta metodología era retrógrada. Los programadores de software no piensan de la misma forma que un usuario a quien está dirigido el software.

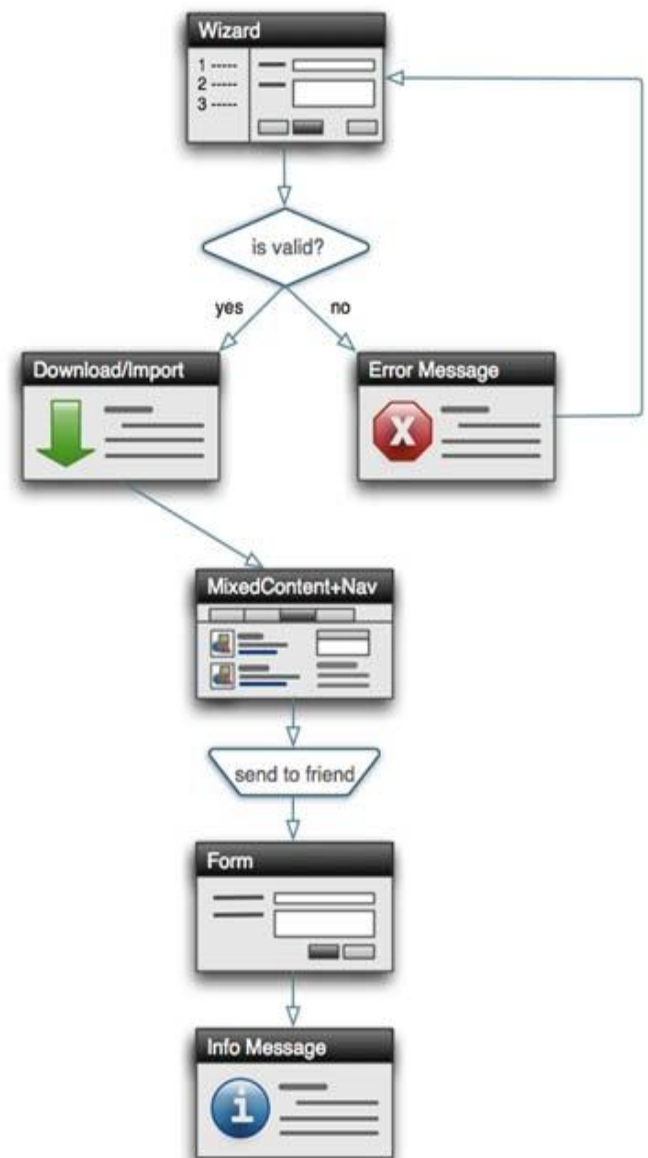
No importa cuanto intentemos ver a través de los ojos del usuario de nuestra aplicación.

Por esta razón gran parte del proceso UXD debe tener lugar antes de comenzar el desarrollo.

Esto comienza con la creación de personajes ficticios que en su vida diaria experimentan alguna clase de dolor que se describe en la historia del usuario.

Los Diagramas de Flujo caracterizan usuarios como robots que siguen un camino predecible cada vez.

La historia del usuario es construida alrededor de un personaje ficticio, lo cual permite a los lectores utilizar su imaginación. Esto ayuda a entender el padecimiento, que el producto de software propone aliviar al personaje de la historia. También infiere en el resultado que aliviará el dolor o problema del



personaje.

Consideración del contexto.

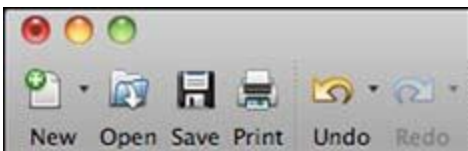
Tres importantes categorías de la experiencia de diseño se apoyan fuertemente en implicancias contextuales:

- Diseño visual e interacción
- Retroalimentación del usuario
- Respuesta y Performance

Diseño visual e interacción.

Consiste en más que elementos visuales tales como el color del esquema, biseles, sombras, y el gusto. Incluye navegación, interacción y referencias metafóricas. A modo de ejemplo una barra de herramientas de una aplicación es el primer lugar donde el usuario busca descifra como acceder a ciertos comandos. Sin importar de que tipo de aplicación se trate el individuo espera ver un set de íconos que le resulten familiares.

Set de standard íconos que usan referencias metafóricas



Los usuarios son considerados como robots que eligen un camino específico para completar una tarea con un programa de software una y otra vez como si estuvieran programados para hacerlo de la misma forma cada vez sin fallar. A esto lo llamaremos el

camino feliz. No se pueden predecir los caminos que el individuo tomará para alcanzar su meta, y por lo tanto su trabajo consiste en requerir el menor pensamiento posible por parte de la persona que trabaja en el software. Esto reduce en gran medida las chances de elegir un camino oscuro, el cual puede llevar a un callejón sin salida. Aunque el diseño en la figura 20.3 no es muy impresionante, tiene alta puntuación en términos de la experiencia del usuario. Esto es porque emplea características como paginación de series de datos de gran tamaño, una actualización automática de Facebook, y navegación de estado por pestañas. Estas características son valiosas, pero uno de los elementos más críticos para proveer una experiencia agradable es la retroalimentación.



Retroalimentación del usuario.

La retroalimentación está relacionada con el público para el cual se está construyendo. Por ejemplo, a no ser que esté escribiendo una aplicación para un profesional IT, probablemente no sería considerado amigable para el usuario si mostrara el seguimiento de la pila AMF en una alerta si la aplicación presenta un error del servidor. Del mismo modo, utilizar notificaciones como "Un error desconocido ha ocurrido" hará que el usuario se sienta inútil y frustrado. Por el bien de las buenas prácticas, usted debe proveer un mensaje de error informativo al usuario y darle algunas directivas de que debe hacer a continuación. Usualmente es una buena idea volcar seguimientos de pila en un archivo de registro cuando se produce un error. De este modo, si la sugerencia es contactar al soporte técnico, el centro de ayuda puede ayudar al usuario a localizar el archivo de registro y enviarlo al centro de ayuda para la resolución de problemas. Todo este trabajo da sus frutos cuando nos conduce al arreglo de un error y la liberación de un parche que salva a miles de usuarios de verse frustrados por el mismo error. Esto también hace que la compañía ahorre dinero en las llamadas a soporte técnico y previene que se genere una pérdida de confianza en el producto.

Respuesta y Performance.

Una interfaz de Flex cuando se combina con los efectos animados, puede llegar rápidamente a un interbloqueo (deadlock), en última instancia, salir de que deje de responder irá en función de la gravedad de la situación.

En Aplicaciones Enriquecidas de Internet altamente personalizadas, desarrolladas en Flex, no es raro encontrarse en una situación en la que puede ser necesaria la creatividad para superar los obstáculos de rendimiento o performance. La solución es adoptar un enfoque de mejores prácticas para acercarse a Flex. He creado un acrónimo que yo llamo el modelo VIBE como una manera de simplificar el proceso de desarrollo de aplicaciones que se centran en las historias de usuario en oposición a los medios tradicionales tales como el modelado UML y los complejos diagramas de flujo.

El modelo VIBE.

Yo uso el modelo VIBE como una forma de medir la experiencia del usuario de un producto de software. VIBE es un acrónimo de :

- * Visual appeal (Atractivo visual)
- * Interactive experience (Experiencia Interactiva)
- * Bussines Optimazition (Optimización del Negocio)
- * Extensibility (Extensibilidad)

Definición de la palabra VIBE es : "El estado emocional de una persona o de la atmósfera de un lugar según lo comunicado y sentido por los demás."

Como se puede ver, la palabra que se utiliza para el acrónimo es apropiado. La frase clave que se utiliza en la definición es "estado emocional".

Al evaluar la primera pregunta, debería ser : "¿Cuál es la sensación que percibo al utilizar este producto ?

La respuesta siempre debe limitarse a frases cortas que sólo describen sentimientos y emociones. Por ejemplo, "frustrado y molesto" califica como una respuesta válida a la pregunta,

mientras que "torpe y lento" expresa una opinión de la funcionalidad del software (o la falta de la misma) que es la causa de un estado emocional, que el individuo falla al describir el mismo. Su objetivo en este primer paso es centrarse en el efecto, no en la causa. En la segunda etapa de la evaluación, se puede determinar la causa del estado emocional iterando a través de los cuatro elementos del modelo VIBE.

Utilización del modelo VIBE ; Vamos a empezar con la parte más visual de la aplicación: temas, skins (mascara), CSS y efectos.

Los Skins son una serie de elementos gráficos que, al aplicarse sobre un determinado software o aplicación, modifican su apariencia externa.

Visual appeal.

El diseño visual de una aplicación Flex es creado con el uso de temas, skins (máscara), CSS y efectos. Una aplicación puede utilizar cualquier combinación de estos cuatro elementos de diseño. Como desarrolladores de Aplicaciones Enriquecidas de Internet, utilizamos estas herramientas para mejorar el atractivo visual de estas aplicaciones.

Usando y creando temas.

Un tema describe el diseño general de componentes para una aplicación Flex específica. Los temas son empaquetados en la forma de un archivo SWC. El tema por defecto de las aplicaciones Flex 3 se conoce como Halo, mientras que el tema por defecto para Flex 4 se conoce como Spark.

El tema Spark es una combinación de estilos y clases skins de programación que construyen la apariencia y la arquitectura de los componentes que están contenidos en el paquete Spark. La alternativa es no compilar los objetos que se poseen en un archivo SWC y utilizar el tema como un conjunto disperso de archivos.

Tabla: Ventajas y desventajas de utilizar un archivo SWC contra objetos que se poseen independientes.

Temas compilados con SWC	<ul style="list-style-type: none">- Más fácil de distribuir.- Reduce el tiempo de compilación (ya xzprecompiled).	<ul style="list-style-type: none">- La encapsulación hace los cambios de diseño un tanto tediosos (se debe volver a compilar).- Más difícil de depurar los problemas de visualización si la interfaz no funciona correctamente.
Recopilación libre de los archivos	<ul style="list-style-type: none">- El diseño puede ser modificado fácilmente.- Los estilos pueden ser fácilmente inspeccionados con propósitos de la depuración.	<ul style="list-style-type: none">- Prolonga el tiempo de compilación de la aplicación.- Riesgo de pérdidas de activos rompiendo espacios de trabajo locales de los desarrolladores.

De esta forma, los desarrolladores no tienen que perder tiempo modificando los objetos que

poseen y recompilar por separado los SWC del tema, que es independiente de la aplicación.

Aplicación de un tema.

Un tema que se aplica mediante el argumento compilador de temas. Si va a utilizar Flash Builder, se puede acceder a los argumentos del compilador navegando a Proyecto > Propiedades > Flex Compiler y localice el campo llamado Argumentos Adicionales del Compilador, como se muestra en la siguiente figura.

Si no va a utilizar Flash Builder y en cambio se utiliza el compilador de línea de comandos, establezca el tema con el argumento correspondiente en la línea de comando, al igual que :

```
mxmclc -theme /Users/path/to/Theme.swc /Users/path/Main.mxml
```

Observe que el primer argumento que sigue a la declaración tema es la ruta de acceso al tema de SWC, seguido de un espacio y luego la ruta al archivo de la aplicación principal a la que se debe aplicar el tema.

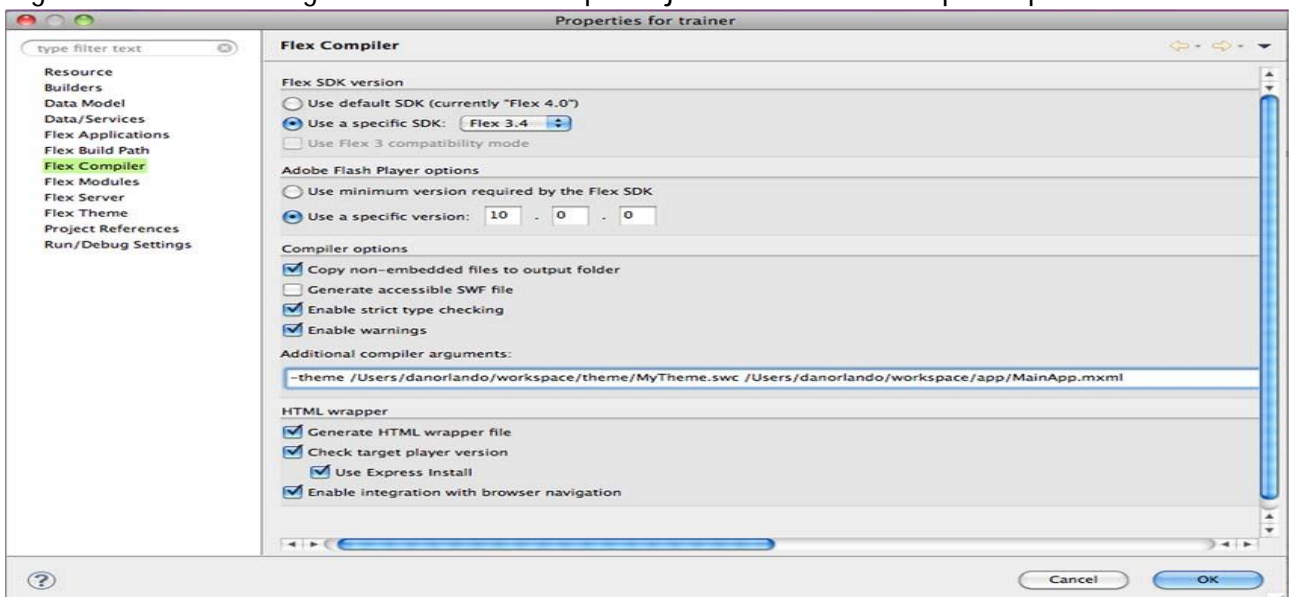
Creando un tema SWC.

Un archivo de tema SWC contiene todas las sub-partes que pertenecen al tema y es muy similar a la creación de un SWC Runtime Shared Library (RSL) o SWC de componente personalizado porque usted usa `compc` en la línea de comandos o construye a partir de un proyecto de biblioteca, si está utilizando Flash Builder. Por lo general cuatro tipos de archivos se compilan en un tema de SWC :

- * CSS hojas de estilo
- * Archivos media (imágenes, videos, SWFs)
- * Fuentes
- * Clases de capas programadas (Archivos ActionScript)

El método preferible y más manejable es construir y utilizar un archivo de configuración para compilar su tema SWC. Este es un archivo XML que especifica todos los archivos que usted quiere incluir en el SWC.

Figura : El tema del argumento es utilizado para fijar el tema en el campo etiquetado.



Argumentos de Compilador Adicional.

El archivo de configuración XML se ubica en la raíz del directorio de la aplicación, la cual por defecto es llamada src. El archivo de configuración XML contiene una lista de archivos que usted quiere que estén incluidos en el SWC.

Un archivo config puede hacer las opciones de compilación más fáciles de leer y mantener.

```
<?xml version="1.0"?>
<flex-config xmlns="http://www.adobe.com/2006/flex-config">
<output>MyTheme.swc</output> (Definición de salida archivo SWC resultante)
<include-file> (Para CSS y archivos media)
<name>mycss.css</name>
<path>c:/myfiles/themes/mycss.css</path>
</include-file>
<include-file>
<name>uplcon.jpg</name>
<path>c:/myfiles/themes/assets/uplcon.jpg</path>
</include-file>
<include-file>
<name>downlcon.jpg</name>
<path>c:/myfiles/themes/assets/downlcon.jpg</path>
</include-file>
<include-file>
<name>overlcon.jpg</name>
<path>c:/myfiles/themes/assets/overlcon.jpg</path>
</include-file>
<include-classes> (Para capas programadas)
<class>MyButtonSkin</class>
<class>MyAccordionHeaderSkin</class>
<class>MyControlBarSkin</class> (La raíz del nodo debe ser flex-config)
</include-classes>
</flex-config>
```

Dando estilo a las aplicaciones Flex 4 con CSS.

Se puede utilizar componentes de skinning o CSS para lograr lo mismo. En algunos casos es más ventajoso usar los dos. Se debe asegurar de no hacer difícil para otros desarrolladores el seguimiento de su código porque usted utiliza los dos métodos. Cuando utilizo CSS con mis aplicaciones Flex 4, particularmente organizar las cosas de forma tal que uso CSS específicamente para dar estilo y clases de componentes skin MXML para manejar el layout. Mi preferencia personal es usualmente escribir las clases componentes en ActionScript puro (con las capas en MXML). Esto me deja con hojas de estilo CSS, capas MXML, y clases componentes ActionScript.

CSS sirve con un propósito un tanto diferente en Flex 4, aunque se han hecho mejoras significativas en el uso de hojas de estilo para Flex 4 (aunque este no es necesariamente el modo recomendado de hacer las cosas), mayormente como el resultado de los pedidos de la comunidad. Estas mejoras incluyen :

- * Selectores globales por nombre de espacio
- * Selectores ID
- * Selección descendente
- * Selección por estado de componentes (pseudo selectores)
- * Selectores de clase múltiple

Veamos estos temas :

Selectores globales por nombre de espacio.

CSS 3.0 incluye soporte para nombres de espacios dentro de las hojas de estilo. Cuando se combina la división de la estructura de Flex entre Spark y MX, tiene sentido construir el soporte de nombres de espacio para CSS en Flex 4. El uso de nombres de espacios con CSS en Flex 4 no es una característica opcional; es una exigencia. Si usted no declara los nombres de espacio antes de empezar a hacer las declaraciones de estilo, usted será alertado con errores de compilación cuando intente construir su proyecto. Como se muestra en la imagen a continuación, los nombres de espacio son declarados al principio del archivo.

Listing 20.2 Namespace declaration and usage with CSS in Flex 4

```

@namespace s "library://ns.adobe.com/flex/spark";
@namespace mx "library://ns.adobe.com/flex/mx";

mx|Button {
    baseColor:#FFF;
    color:#000;
}

s|Button {
    baseColor:#000
    color:#FFF;
}

```

Annotations:

- 1 Declare the Spark namespace
- 2 Declare the MX namespace
- 3 Notation for an MX button
- 4 Notation for a Spark button

Además, estos nombres de espacio deben también ser lo primero que se declare después de la etiqueta <fx:Style/> en MXML. Declarar los nombres de espacio no tiene utilidad a no ser que se haga referencia a ellos en el código siguiente de la aplicación.

Con CSS en Flex 4, los prefijos de los nombres de espacio se deben situar en el frente de su selector de estilo con un delimitador pipe, como se muestra en la figura la Declaración de nombres de espacio y uso de CSS en Flex4

Como puede ver en la figura, los nombres de espacio son declarados en las hojas de estilo utilizando @nombresdeespacio antes de que ningún estilo sea escrito. En el código de muestra, el nombre de espacio de Spark es declarado primero B, después seguido por el nombre de espacio MX C. Dos selectores son nombrados después para que los estilos puedan ser aplicados a ellos, empezando con el componente botón MX D y seguido por el componente botón Spark E. Aunque botón es un selector global, éste es único en el nombre de espacio respectivo en Flex 4, como se especifica en el lado izquierdo del delimitador pipe.

Selectores ID.

Muchos desarrolladores encuentran la funcionalidad de CSS en Flex 3 limitada. Una de las mayores quejas era que la implementación del CSS en Flex 3 no permitía asignar estilos haciendo referencia a la propiedad ID de un objeto de visualización.

En la implementación de la hoja de estilo en Flex, CSS es incorrecta porque falta la parte de

“cascada”. Poniéndolo simple, usted no define el layout específico de los componentes en el escenario con CSS. En lugar de esto, se hace en una vista de clases de una aplicación MXML o en una clase de capa de un componente MXML en Flex 4.

Por ejemplo, digamos que usted crea un botón de Spark con un id de botón de presentación y quiere que el color base del botón sea negro y el texto blanco. Para lograr esto, usted utiliza el siguiente código:

```
s|#submitButton {  
baseColor: #000000;  
color: #FFFFFF;  
}
```

Note en el código que el símbolo numeral precede el selector id, que es lo mismo que en CSS para HTML / XHTML. Podría llevar las cosas un paso más allá y limitar estos estilos para ser aplicados solamente a botones de Spark que tienen un id de #submitButton, lo cual es útil si, por ejemplo, usted tiene un componente LinkButton en otro archivo MXML con el mismo id. Es una práctica mejor cambiar el id si usted tiene un conflicto, pero por lo menos ahora usted sabe que hacer si eso no es una opción.

El código para esto es el siguiente:

```
s|Button#submitButton {  
baseColor: #000000;  
color: #FFFFFF;  
}
```

Los Selectores ID son un gran agregado a CSS para Flex, pero no termina ahí.

Selección descendente.

En ella referencia un objeto de visualización particular solamente cuando existe el objeto de visualización como un hijo de un contenedor particular. Por ejemplo, si desea que todos los componentes de Spark button que se encuentran en el interior de un contenedor SparkPanel tenga texto rojo, tendrá que utilizar el siguiente código en su hoja de estilo:

```
s|Panel s|Button {  
color: #FF00000;  
}
```

Note que en este código el selector s|Button viene después del selector contenedor padre, separado solamente por un espacio. Usted puede llevar las cosas al siguiente nivel nuevamente agregando la condición de aplicar el estilo solo cuando exista el Sparkbutton dentro de un SparkPanel con un id myPanel. En este caso, el código se modifica para agregar el selector id al padre, como se muestra aquí:

```
s|Panel#myPanel s|Button {  
color: #FF00000;  
}
```

Selección por estado del componente (PSEUDO SELECTORES).

Esto especifica las pre-condiciones a aplicar al estilo solo cuando el componente especificado está en un estado particular. El código sería como sigue :

```
s|Button:up {
```

```
baseColor: #000000;  
color: #FFFFFF;  
}
```

En este ejemplo de código, dos puntos separan el selector y el nombre del estado, al cual se debe aplicar el estilo. Flex 4 provee mucha mayor flexibilidad en los nombre de estado. Digamos por ejemplo, que usted tiene una clase de capa para un componente de un botón que incluye el siguiente código para especificar los estados del componente:

```
<s:states>  
<s:State name="enabled" />  
<s:State name="focus" />  
<s:State name="selected" />  
</s:states>
```

Como usted puede ver en el código de estilo de la siguiente figura, que se corresponde con los estados especificados aquí, su botón aún funciona como usted espera, aunque no está utilizando los nombres de estado arriba, sobre y abajo.

Listing 20.3 CSS state selector example

```
@namespace s "library://ns.adobe.com/flex/spark";  
  
s|Button:enabled {  
    baseColor: #5387B7;  
    color: #304F6B;  
}  
  
s|Button:focus {  
    baseColor: #917541;  
    color: #B8A553;  
}  
  
    baseColor: #B8A553;  
    color: #6B5630;  
}
```

Corresponds to
the enabled state

Corresponds to
the focus state

Corresponds to

Figura : Ejemplo de selector de estado CSS.

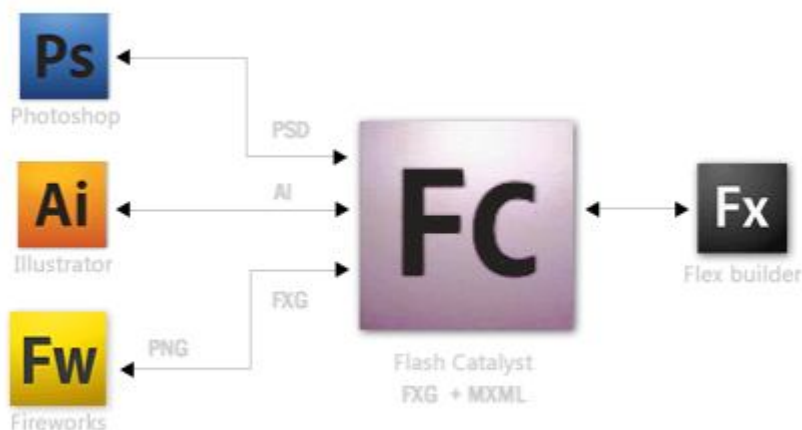
Selectores de clase múltiples.

Esta mejora le permite especificar más de un selector de clase al establecer la propiedad de nombre de estilo en un componente. Digamos por ejemplo, que usted construye un componente personalizado que usted quiere que utilice dos clases de selectores en su hoja de estilo, cuando los estilos ya existen, aunque en dos clases diferentes de selectores. Puede establecer el nombre de estilo como propiedad del componente como sigue :

```
styleName="fontStyle headerStyle"
```

Optimización del flujo de desarrollo con Flash Catalyst

Flash Catalyst CS5 le permite importar diseños de interfaz de usuario creados con Photoshop, Illustrator y Fireworks y añadir interactividad, estados, transiciones y efectos a través de una interfaz de usuario intuitiva que no requiere ningún tipo de programación.



Hasta el momento, Adobe Illustrator CS5 es la única aplicación capaz de "ir y volver" con Flash Catalyst CS5. Esto significa que se puede crear un archivo de Illustrator CS5, importarla a Catalyst, y luego exportar de Catalyst y reimportación en Illustrator si necesita realizar cambios adicionales de Illustrator.

Flash Catalyst CS5 puede abrir FXG archivos exportados desde cualquiera de los programas de diseño CS5

Flash Catalyst CS5 tiene la capacidad de exportar directamente un solo archivo FXP (Proyecto de Flex), que puede entonces ser perfectamente importado en Flash Builder. Flash Builder utiliza FXG para leer y mostrar los gráficos.

La capa de negocio de la aplicación, o la lógica de comportamiento, se agrega al proyecto de Flash Builder, y luego la depuración y perfilado puede iniciarse antes de exportar la versión final.

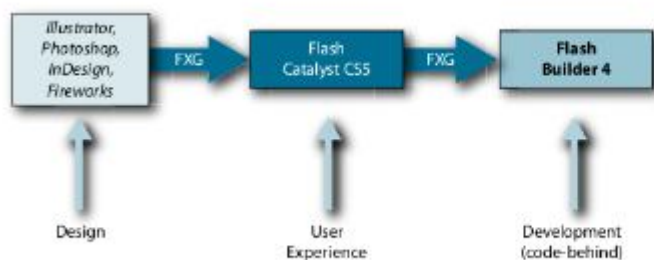


Figure 20.5 Moving from left to right, notice that FXG is used to translate graphics between the Adobe applications.

DEGRAFA:

Un marco declarativo gráficos disponibles para Flex ha estado disponible desde hace varios años,

llamado Degrafa. Este marco de código abierto está activo y seguirá siendo porque el propósito y la dirección de Degrafa son diferentes de la de FXG.

Probablemente la cosa más atractiva sobre Degrafa es lo fácil que es.

El marco Degrafa también es maduro, y el hecho de que es de código abierto ciertamente no hace daño

Su nombre viene de “Declarative Graphics Framework” y es eso, un framework para generar gráficos a partir de declaraciones, utilizando para ello, no podía ser de otra forma, [MXML](#). (Macromedia eXtensible Markup Language)

Mejorar la experiencia con los efectos

No hay duda de que la animación es siempre un divertido tema para hablar de la hora de construir aplicaciones ricas de Internet y Flex 4 no defrauda cuando se trata de transiciones animadas y efectos. Flex 4 presenta una API más sofisticado para la producción de efectos que el de la versión 3. La nueva API se aprovecha de las características que se incluyeron con la versión de Flash Player 10, como simulación de 3D .

3D PRIMER EFECTO

Es probable que los efectos más llaman la atención, imponente y agradable a la vista son las que simulan el uso del espacio 3D. La versión de Flash Player 10 introdujo algunas capacidades 3D simulado convincentes añadiendo un eje Z en el escenario. Esto es especialmente interesante para los desarrolladores de Flex y AIR, debido al impacto que las visualizaciones 3D pueden tener en la experiencia de los usuarios.

Listado 20,4 demuestra lo fácil que es utilizar el efecto Rotar3D en Flex 4.

Listado 20,4

Rotación 3D en el eje X con el efecto Rotar3D

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application name="Spark_Rotate3D_test"
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/mx">
  <s:layout>
    <s:BasicLayout />
  </s:layout>

  <fx:Declarations>
    <s:Rotate3D id="rotate3DX"
      target="{image}"
      angleXFrom="0"
      angleXTo="360"
      duration="2000"
      autoCenterTransform="true"
    />

    <mx:ApplicationControlBar width="100%" cornerRadius="0">
      <s:Button id="buttonX"
        label="Rotate3D X-axis"
        click="rotate3DX.play();" />
    </mx:ApplicationControlBar>

    <s:BitmapImage id="image"
      source="@Embed('assets/fx_appicon.jpg')"
      resizeMode="scale"
      smooth="true"
      horizontalCenter="0"
      verticalCenter="0"
      width="100"
      height="100" />
  </fx:Declarations>
</s:Application>
```

Place effects inside
Declarations tag

1 Declare values
and bindings

1 Declare values
and bindings

2 Effect triggered
using play ()

Como se muestra en el listado 20.4, se empieza por declarar una instancia del objeto Rotar3D Spark dentro de las `<fx:Declarations />` tag B. A continuación, se enlaza el efecto a un objeto en la lista de visualización. Enlazar el efecto a un objeto de visualización específico es opcional. No vinculante que permite el efecto que se utiliza en más de un objeto llamando `rotate3DX.play(target)`, donde destino es cualquiera de los objetos de la lista de visualización que pueden ser animados.

En el ejemplo, `angleXFrom` y `angleXTo` se utilizan para rotar el objeto. La propiedad `angleXFrom` se utiliza como el punto de partida de la rotación, mientras que `angleXTo` se usa como el punto final de la rotación.

La propiedad `duration` es la longitud por la cual la animación debe durar. En el ejemplo, hemos creado la duración a 2000 milisegundos, haciendo que el objeto de tomar 2 segundos para completar una sola rotación. La propiedad `autoCenterTransform` dice Flex que queremos que el objeto de utilizar su centro como punto de registro de la rotación, en lugar de utilizar su punto de píxel superior izquierdo como lo haría normalmente.

Lo más importante, el efecto se activa en el tecleo de buttonX como se indica con el código `click = "rotate3DX.play ()"`. Tenga en cuenta que nosotros no tuvimos que pasar el objeto al constructor del método de reproducción del efecto porque el efecto ya está vinculada a un objetivo 2.

Ejemplos: <http://blog.flexexamples.com/>

Optimización de negocio

- TDD
- Pruebas unitarias
- Escalabilidad
- Extensibilidad
- Código limpio
- Convenciones o nomenclaturas prácticas

Efectos

Un efecto modifica las propiedades visuales de un componente durante un período de tiempo. Estas propiedades incluyen la posición de un componente, la transparencia, tamaño, y más. Para efectos de imagen se puede mirar a su alrededor.

Piense en un péndulo de un reloj de péndulo balanceándose de izquierda a derecha, el desvanecimiento de TV en cuando se enciende, o un paisaje a través de una lente de la cámara como el zoom dentro y fuera. Esos son todos los ejemplos de efectos. En los tiempos prehistóricos (hace dos años en Flex 3), se puede animar sólo los objetos de visualización (componentes de interfaz de usuario), pero Flex 4 le permite apuntar a objetos / componentes, objetos gráficos (entradas degradado y así sucesivamente), y los filtros de visualización. Por ejemplo, se puede pasar a la transparencia (alfa) hacia arriba o hacia abajo durante un período de cinco segundos para crear un efecto de fundido.

En esta sección usted aprenderá lo que está disponible para usted como desarrollador de Flex a través de los diferentes efectos y combinaciones de efectos. Para profundizar en el que tenemos que mirar primero en las capacidades de efectos predeterminados que se proporcionan en Flex 4.

Los efectos disponibles

Flex viene con una variedad de efectos listos para salir de la caja. La Tabla 21.1 muestra los efectos prediseñados Flex 4.

Table 21.1 Available Flex effects out of the box

Effect	Animates
Fade	alpha property
Move and Move3D	x, y, and/or z properties
Resize	Width and height
Scale and Scale3D	scaleX, scaleY, and/or scaleZ properties
Rotate and Rotate3D	rotation, rotationX, rotationY, and/or rotationZ properties
CrossFade	Two objects or the text of an object between states
Wipe	Two bitmaps

Esta lista incluye algunas de las habilidades que son pre-compilados y están disponibles para usted en este momento.

¿Qué es eso? Al no ver los efectos que una vez vio en Flex 3? Las cosas han cambiado un poco. Tabla 21.1 es simplemente una lista de efectos predefinidos, pero Flex 4 te ofrece clases de base de proxenetismo sus propios efectos. Animaciones mucho más avanzadas son posibles MotionPaths utilizando, fotogramas clave y numerosas combinaciones de efectos, filtros y shaders. Los efectos de la tabla 21.1 se extienden las clases de efectos principales. Tabla 21.2 muestra las clases de efectos principales y lo que hacen.

Junto a estos efectos predefinidos también se pueden combinar efectos, creando lo que se conoce como efectos de composición.

Table 21.2 Core effect classes

Effect	Animates
Animate	Base class to all effects; allows granular customizations
AnimateColor	A color over time using interpolation between two colors on a per-channel basis
AnimateFilter	Filters (BlurFilter, GlowFilter, and so on)
AnimateTransformShader	Two bitmaps through a pixel-shaders program (CrossFade and Wipe extend this class.)
AnimateTransform and AnimateTransform3D	A combination of translation, scale, and rotation properties (Move, Scale, Rotate and their 3D counterparts extend this class.)

Efectos compuestos

Para hacer cosas divertidas, puede combinar efectos usando cualquiera de estos métodos:

- Secuencia-Reproduce un efecto tras otro
- Paralela-Reproduce todos los efectos en paralelo, a la vez

Uso de los efectos

Vamos a ver cómo podemos utilizar estos efectos, a partir de los mecanismos de activación de efecto y luego saltar en ejemplos de cada uno.

Causa y efecto

Los efectos se reproducen desde un disparador. Piense en ello como causa y efecto en la vida normal. Usted apreta un botón y arranca tu coche, acciona un interruptor y las luces se encienden y así sucesivamente. Lo mismo puede decirse de los efectos de programación. Un usuario hace algo y usted, el desarrollador, responder a ese gatillo, jugando con un efecto. Vamos a explorar las diferentes disparadores usados para integrar efectos.

Hay tres factores desencadenantes del efecto en Flex:

- Event-El efecto es expulsado automáticamente cuando un evento determinado efecto específico ocurre (por ejemplo, `<s:Button mouseOverEffect="myEffect" />`).
- Mediante programación, puede crear un efecto de declaración o sobre la marcha y ejecutarlo como se desee (por ejemplo, `myEffect.play ()`).
- Transiciones-Se ejecuta durante un cambio de estado.

El efecto puede utilizar cualquiera de estos métodos. Cómo o cuando se utilicen dependerán de su situación específica. En el futuro, vamos a explorar cada uno de los factores desencadenantes de efectos y ver ejemplos de cómo usarlos.

Disparado por Eventos:

Como mencionamos en la sección anterior, un efecto activado por eventos se produce como resultado del disparo de un evento específico. Todos los componentes visuales se apoyan en una serie de propiedades, lo que sirve para especificar los efectos que desea reproducir de forma automática en el evento correspondiente.

TABLA

Propiedad	Efecto
addedEffect	El componente que se añade a un recipiente
creationCompleteEffect	El componente fue creado
focusInEffect	El teclado se centra en el componente
focusOutEffect	El teclado centrándose fuera el componente
hideEffect	Propiedad visible del componente variable en false
mouseDownEffect	El botón del ratón está presionado sobre el componente
mouseUpEffect	El botón del ratón deja de ser presionado en el componente
moveEffect	El cambio de posición de componentes
removedEffect	El componente se retira del recipiente en el que estaba
resizeEffect	El componente que se está cambiando
rollOutEffect	La mudanza del ratón de la componente
rollOverEffect	La mudanza del ratón sobre el componente

SUGERENCIA sus propios componentes personalizados pueden exponer a efecto desencadena también. Este capítulo no comprende la escritura de sus propios efectos personalizados, pero se

puede lograr a través de los metadatos de este modo: [Effect (name = "eventNameEffect",event = "eventName")].

Vamos a utilizar esta información en un ejemplo práctico. Un efecto popular para álbumes de fotos es para que la imagen crezca en tamaño, pasa el ratón sobre el tema; 21,1 lista hace con un efecto de cambio de tamaño en un Rect negro en un grupo.

Listing 21.1 Using the Resize effect to enlarge an image on mouse-over

```
<?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">
  <fx:Declarations>
    <s:Resize id="fxEnlarge" widthTo="300" heightTo="300"
              library://ns.adobe.com/flex/spark/>
```

1 Define enlarge
resize to
300x300

Using effects

473

```
    <s:Resize id="fxShrink" widthTo="100" heightTo="100"/>
  </fx:Declarations>
  <s:Group width="200" height="200"
          rolloverEffect="{fxEnlarge}" rollOutEffect="{fxShrink}">
    <s:Rect id="box" width="100%" height="100%">
      <s:fill>
        <s:SolidColor color="black" />
      </s:fill>
    </s:Rect>
  </s:Group>
</s:Application>
```

2 Define shrink
resize to 100x100

3 Set the effects

¿Podría ser más fácil? El Grupo aplica dos efectos, rolloverEffect D y roll-OutEffect D, cuando el ratón se mueve sobre y fuera del gráfico, tal como se muestra en la figura

21.1. Cuando se distribuye el evento rollover, el efecto fxEnlarge B se le dice que play (), y lo mismo es cierto para fxShrink C y la rollOutEffect.

NOTAS - Los efectos van en un bloque de declaraciones porque no son componentes visuales.

NOTA - Al coincidir con un disparo con efecto, se conoce como un comportamiento.

Efectos que se activan son fáciles de crear, sino que se limitan a los desencadenantes conocidos apoyados por componentes.

Programación aplicar un efecto

Usted acaba de aprender cómo que usted puede aprovechar los acontecimientos desencadenados. Esto significa que toma un caso y asigna un efecto a él, y luego automáticamente el efecto juegue. Su aplicación puede incluir escenarios únicos, a los que es posible que desee aplicar un efecto a su elección en el tiempo y la forma que elija para su

aplicación. De este modo, se puede aprender a jugar con efectos de programación.

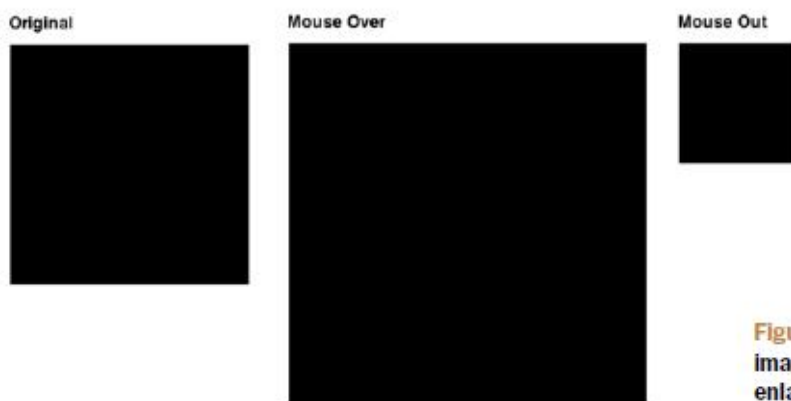


Figure 21.1 Mousing over the image triggers the image to enlarge using the **Resize** effect.

EFFECTOS DE ACTIVACIÓN CON ACTIONSCRIPT

Suponga que tiene una función de recordatorio, y si algo se retrasa desea hacer el recordatorio resaltando en rojo. Listado 21.2 muestra este concepto contando los segundos, si se sobrepasa un umbral, el mensaje se ilumina en varias ocasiones.

Listing 21.3 Creating an instance of an effect using pure ActionScript

```
<?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"
               applicationComplete="init()">

  <s:layout>
    <s:VerticalLayout paddingLeft="20" paddingTop="20" />
  </s:layout>
  <fx:Script>
    <![CDATA[
      import flash.utils.Timer;

      import spark.effects.AnimateFilter;
      import spark.effects.animation.MotionPath;
      import spark.effects.animation.RepeatBehavior;
      import spark.effects.animation.SimpleMotionPath;
      import spark.filters.GlowFilter;

      [Bindable]
      protected var secondsTillDue:int = 5;
      protected var _timer:Timer;
      protected var _effect:AnimateFilter;

      protected function init():void{
        var filter:GlowFilter =
          new GlowFilter(0xde7800, 1, 20, 20);
        _effect = new AnimateFilter(box, filter);
        _effect.duration = 1000;
        _effect.repeatCount = 0;
        _effect.repeatBehavior = RepeatBehavior.REVERSE;
        _effect.motionPaths = new <MotionPath>[
          new SimpleMotionPath("alpha", 0, 1)];
        _timer = new Timer(1000); //tick every 1000ms
        _timer.addEventListener('timer', onTimerTick);
        _timer.start();
      }

      protected function onTimerTick(event:TimerEvent):void{
        secondsTillDue = Math.max(secondsTillDue-1, 0);
        if(secondsTillDue == 0){
          _effect.play();
          _timer.stop();
        }
      }
    ]]>
  </fx:Script>

  <s:SolidColor color="black" />
</s:fill>
</s:Rect>
</s:Application>
```

Set GlowFilter to glow orange

Create effect instance

Set up MotionPath for alpha animation

Set timer to tick every 1000ms

Update secondsTillDue value

Recuerde que el uso del ejemplo anterior, donde es relativamente fácil cambiar a un efecto diferente? También puede hacer que el uso de este enfoque, pero la ventaja real de la creación de un efecto de esta forma es la capacidad de crear dinámicamente objetos de efecto. Esto se puede lograr a través de efectos declarativos también, pero con una mezcla de declarativa y programáticas.

Creación de un efecto es grande, pero a veces se quiere un efecto de activar sólo después de otro, o si desea dos efectos para jugar al mismo tiempo sin tener que llamar `play()` de 10 efectos diferentes. Si éste es tu caso, los efectos de compuestos será su nuevo

mejor amigo.

Uso de las transiciones de estado para activar efectos

La última manera de accionar un efecto es a través de una transición de un estado a otro. Listado 21.4 muestra cómo implementar una transición de estado.

Listing 21.4 Implementing a state transition

```
<?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:transitions>
    <s:Transition fromState="*" toState="*">
      <s:Resize target="{box}" />
    </s:Transition>
  </s:transitions>
  <s:HGroup>
    <s:Button label.orange="Black" label.black="Orange">
      <s:click>
        <![CDATA[
          currentState = (currentState ==
            'orange' ? 'black' : 'orange');
        ]]>
      </s:click>
    </s:Button>
  </s:HGroup>
```

1 Define states

2 Define Transition for all states

3 Set desired transition effect

4 Change states on button click

CHAPTER 21 Working with effects

```
<s:Rect id="box" width="200" height="200"
        width.black="150" height.black="150">
  <s:fill>
    <s:SolidColor color.black="#000000"
                  color.orange="#de7800" />
  </s:fill>
</s:Rect>
</s:Application>
```

5 Set dimensions for each state

6 Set color for each state

Listado 21.3 viene directamente desde el capítulo 14 con una modificación: la `<s:transitions/>` código de bloque C. Este bloque define transiciones deseadas para cada estado B. En este caso, cubrir todos los estados con una transición con `*` pero podríamos haber definido las transiciones específicas para cambios de estado específicas. La actual transición es un cambio de tamaño D. Observe que no se da ningún detalle sobre el cambio de tamaño (como `widthTo` o `heightTo`).

Esto es porque la transición se aplica esas propiedades y consecuencia basan en los nuevos valores de estado E para el objeto de destino F.

Usted ha visto algunos ejemplos de los efectos singulares, pero todavía tiene mucho que aprender aquí respecto a los efectos compuestos, se ejecutan múltiples efectos simultáneamente o en secuencia.

Drag & Drop

Cuando Flex salió por primera vez, drag-and-drop (D & D) marcó la pauta, como elemento clave, mostrando lo RIAs basadas en Flex fueron capaces de proporcionar integración con mínimo esfuerzo. Desde la perspectiva del desarrollador, D & D función de Flex es fácil de implementar debido a que muchos componentes nativos proporcionan un mecanismo incorporado para usarlo. En el mismo tiempo, se puede conectar y reemplazar cualquiera de el comportamiento predeterminado en todas las etapas del proceso.

En este libro, queremos mostrar no sólo cómo hacer algo, sino también por que hacer algo.

En las aplicaciones web tradicionales, que se basan en barcos cargados de enlaces y botones a mover los desarrolladores de datos web que no utilizan frameworks de JavaScript modernas (como jQuery) tienden a olvidar el concepto de D & D (en el contexto web), a pesar de que utilizan cada día en el lado de escritorio de la vida. Y debido a que D & D es tan raro en la web, incluso los usuarios olvidan el concepto.

Nosotros proponemos que se consideren D & D desde el punto de vista de facilidad de uso y no sólo porque es posible o por el factor de frialdad. El reto no es tanto el esfuerzo de programación (se verá en este capítulo lo fácil que es), sino más bien la forma de pensar.

Usted querrá proporcionar los paradigmas típicos de gestión de datos comunes en las aplicaciones web, porque eso es lo que los usuarios buscan de manera predeterminada, pero también se debe entrenar a utilizar las capacidades de la aplicación D & D.

En última instancia, el nivel en el cual los usuarios tienen éxito es el nivel en el que su solicitud es aceptada. Llegamos a rodar la pelota para su éxito mediante la revisión del proceso general D & D.

22.1 El proceso de arrastrar y soltar

El proceso de D & D no es más que eventos que se disparó en el distintas etapas del ciclo de vida de arrastrar y soltar.

Primero vamos a obtener algunas definiciones de la forma:

- Drag Initiator- El componente del que se inicia el arrastre.
- Drga proxy-El icono del ratón se muestra al usuario durante el proceso de D & D.
- Drag Source-el arrastre de datos que se transfieren entre el iniciador de arrastre y soltar objetivo. El nombre es un poco engañoso, ya que suena como si fuera la fuente de la operación de arrastre, que es el iniciador de arrastre.
- Drop Target- El componente que recibe el arrastre.

Cuál está en juego en ciertos momentos puede ser confuso, así que vamos a ir a través del proceso de D & D. Incluye una serie de medidas, en función de lo que haga el usuario.

Repasamos cada paso en detalle moderado para que pueda entender lo que el usuario está

haciendo, lo que activa eventos, y lo que el usuario experimenta en cada paso.

1. Se inicia Drag.
 - El usuario hace clic en un componente drag-enabled (ajustando el drag-Propiedad Enabled en true).
 - Evento disparó: mouseDown.
 - Mientras mantiene presionado el botón del ratón sobre el tema, el usuario comienza el proceso de arrastre. El componente de partida es ahora el iniciador de arrastre. Los sucesos activados: mouseMove y dragstart.
2. Se crea Arrastre objeto de origen.
 - Contiene todos los datos asociados con el elemento que se está arrastrando.
3. Se muestra una representación inicial arrastre visual.
 - Un icono rojo (x) indica que el componente del ratón está sobre no se aceptar la caída actual.
 - (+) icono verde se utiliza cuando se permite que la caída actual.
4. Un drag representante destino se muestra como los ratones de usuario a través de los componentes.
 - Un icono rojo (x) indica que el componente del ratón está sobre un componente que no acepta drop.
 - (+) icono verde se utiliza cuando se permite que la caída actual.
 - Los sucesos activados: dragEnter y dragOver.
5. Mouse de un usuario en posible destino de colocación.
 - El proxy de arrastre se actualiza.
 - Evento disparó: dragExit.
6. Usuario deja caer el tema en un objetivo.
 - Drag se ha completado o cancelado.
 - Si el componente aceptó la caída, la resistencia se ha completado.
 - Si el componente rechazó la gota, el arrastre es cancelada y el artículo devuelve (anima de nuevo) a la fuente.
 - Los sucesos activados: dragdrop y dragComplete.

La figura 22.1 muestra un ejemplo de la representación de arrastre con un icono + para mostrar la operación de arrastre se puede completar.

Se trata de un alto nivel de detalle, pero no se preocupe por recordar todo. Caminaremos a través de los pasos de este capítulo para ayudarle a comprender mejor cada concepto.

Como puede ver, los acontecimientos siguen llegando.

Flex es un sistema orientado a eventos. Si hay alguna vez un lugar para eventos, definitivamente, en D & D, y en la siguiente sección explicaremos y aclararemos cada uno de los eventos de D & D.

22.1.1 Eventos de arrastrar y soltar

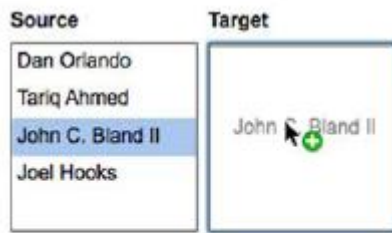


Figure 22.1 The drag proxy provides a visual indicator to the user about whether the operation is allowed.

Tabla 22.1 eventos de arrastrar y soltar

Evento	Descripción	Receptor
mouseDown	iniciador de arrastre	Produce cuando el usuario mouses hace click en el elemento arrastrable. Este es el punto en el que usted puede añadir lógica de negocio para determinar si desea permitir el inicio de una operación de arrastre.
mouseMove	Drag iniciador	Usuario comienza a mover el elemento arrastrable.
dragstart	Drag iniciador	Iniciador de arrastre dispara un evento en sí mismo.
dragEnter	gota objetivo	Drag representación inicialmente pasa sobre una posible destino de colocación. El destino de colocación puede utilizar esto como una oportunidad para examinar los datos y determinar si se acepta o rechaza la entrega. Este evento se dispara varias veces, para cada movimiento del ratón sobre la objetivo. CAVEAT Si está desarrollando para Adobe AIR, este evento se disparará sólo una vez.
destino de colocación dragOver		Se produce inmediatamente después de dragEnter y se envía sólo si la de destino acepta el drop .Se puede utilizar para actualizar visualmente el objetivo de demostrar que está listo para aceptación. CAVEAT Si está desarrollando para Adobe AIR, este evento se disparará independientemente de la aceptación o rechazo del artículo.
dragdrop	gota objetivo	Resultado del usuario suelta el botón del ratón. Aquí es donde se comprometer los datos y finalizar la operación. Destino de colocación
dragExit		Se produce si el usuario no suelte el botón del ratón y se aleja el destino de colocación, que van desde dragOver a dragExit. Aquí es donde deshacer / restablecer cosas que hiciste en el proceso dragOver. iniciador de arrastre
dragComplete		Enviado al iniciador de arrastre cuando el proceso de D & D termina por cualquier razón (el usuario suelta el artículo o cancela la operación al permitir ir el botón del ratón mientras no aceptar más de un componente).

22.2 Implementación de arrastrar y soltar en componentes de Flex

Ahora tiene una familiaridad con el proceso de D & D. Puede parecer mucho para recordar o de perder de vista, pero Flex hace super simple con un conjunto de componentes que poseen nativos D & D.

Cubrimos los componentes que tienen D & D, como agregar D & D de apoyo, y las operaciones diferentes de D & D (copiar, mover, la clasificación interna de varios artículos D & D y bidireccionales D & D) antes de pasar a la aplicación de D & D en sus propios componentes personalizados.

22.2.1 Componentes con soporte drag-and-drop nativa

Los candidatos obvios de uso para las operaciones de D & D son componentes de lista de datos. En Flex, esos son los componentes basados en List, en particular los siguientes, que se muestran en la tabla 22.2.

Te habrás dado cuenta de la profundidad de los componentes de Halo frente a componentes Spark.

Recuerde, Spark aún no 1:1 con Halo, en términos de componentes disponibles.

También

Component	Namespace
List	Spark, Halo
TileList	Halo
HorizontalList	Halo
Tree	Halo
DataGrid	Halo
AdvancedDataGrid	Halo
PrintDataGrid	Halo

Table 22.2 Available D&D components

Recuerde que la lista es un componente de gran alcance. Usted puede construir un TileList o HorizontalList en Spark ajustando el diseño de TileLayout o HorizontalLayout, respectivamente. Como el conjunto de componentes Spark continúa creciendo, D más nativo + D Spark componentes estarán disponibles. En las próximas secciones vamos a centrar exclusivamente en la Lista de mantener nuestras mentes en Flex 4. Más adelante en el capítulo cubriremos cómo D & D entre el Spark y los componentes de Halo.

Sabiendo que los componentes a utilizar son grandes, pero aprender cómo usarlos es otra cosa. Vamos a saltar con entusiasmo y aprender cómo habilitar D & D en un componente List.

22.2.2 Habilitación de D & D en las listas

El componente List soporta de forma nativa D & D con el valor de las propiedades específicas. Habilitación de D & D es fácil. El componente List admite dos propiedades para que esto suceda (véase el cuadro 22.3).

Table 22.3 Drag-and-drop properties of the List component

Property	Type	Description
dragEnabled	Boolean	Indicates whether the component is allowed to serve as a drag initiator. true or false (default).
dropEnabled	Boolean	Indicates whether the component is allowed to serve as a drop target. true or false (default).

NOTE There's a third property, `dragMoveEnabled`, but you don't need to worry about it until section 22.2.3.

Using those two properties, listing 22.1 shows the code used in figure 22.1 (D&D process).

Listing 22.1 List accepting dragging and dropping from another List

```
<?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">
  <s:layout>
    <s:HorizontalLayout paddingLeft="20" paddingTop="20" />

  </s:layout>
  <fx:Declarations>
    <s:ArrayCollection id="authors">
      <fx:Object label="Dan Orlando"
                 telephone="555-0000" />
      <fx:Object label="Tariq Ahmed"
                 telephone="555-0001" />
      <fx:Object label="John C. Bland II"
                 telephone="555-0010" />
      <fx:Object label="Joel Hooks"
                 telephone="555-0011" />
    </s:ArrayCollection>
  </fx:Declarations>
  <s:VGroup>
    <s:Label fontWeight="bold" text="Source" />
    <s:List dragEnabled="true" dataProvider="{authors}" />
  </s:VGroup>
  <s:VGroup>
    <s:Label fontWeight="bold" text="Target" />
    <s:List dropEnabled="true"
            dataProvider="{new ArrayCollection()}" />
  </s:VGroup>
</s:Application>
```

Define author data

Set source's dragEnabled to true

Set target's dropEnabled to true

La figura 22.1 muestra el resultado de una operación de D & D usando listado 22.1. Tenga en cuenta que lo único especial del listado 22.1 es el uso de `dragEnabled` y `dropEnabled` en las dos Listas, respectivamente. Véase la tabla 22.3 para recordar su uso. En este capítulo se hablará más acerca de la personalización y control, pero esta lista es material de aplicación en el mundo real envuelto en un ejemplo sencillo.

TIP Flex no restringirá los elementos duplicados se eliminen de la lista.

Usted puede agregar esta funcionalidad al interceptar el evento `DragEnter` y aceptar o negar el artículo basado en su propia solicitud de datos (por ejemplo, sin duplicados).

De forma predeterminada, la operación de D & D se realizará una copia. Una copia es

simplemente tomando una copia de el elemento de arrastre y agregarlo a otro componente sin afectar el componente fuente.

Flex también le permite moverse con facilidad el tema, quitar el elemento de la fuente, y añadirlo a su destino.

22.2.3 Movimiento contra copia

Usted puede haber notado que la inclusión de 22,1 resultados en una operación de copia en vez de mover los datos de una lista a otra. Si quieres hacer una operación de movimiento en lugar de una copia, el cambio es simple.

Componentes basados en List admiten una propiedad `dragMoveEnabled`, que por defecto es establecido en `false` y hace que el comportamiento por defecto de la copia de la partida de datos. HALO TIP Hay una excepción: El componente Tree dispone lo contrario predeterminado. Todo lo que necesitas hacer es establecer `dragMoveEnabled` en `true` y el D & D realiza un movimiento en lugar de una copia. El siguiente código se basa en la oferta 22.1, pero por razones de brevedad nos muestran sólo los cambios que permitan una operación de movimiento:

```
<s:List dragEnabled="true" dragMoveEnabled="true"
        dataProvider="{authors}" />
```

Eso es todo lo que se necesita para permitir movimiento. Tenga en cuenta que `dragMoveEnabled` se añadió a la lista, que cuenta la lista de fuentes ", cuando el evento `dragComplete` ocurre, elimine el elemento de arrastre de mi `dataProvider`. "Figura 22.2 muestra el resultado de dos operaciones de D & D basados en anuncios 22.1 más los cambios fragmento anterior.

Este flujo de trabajo es probablemente algo con lo que está muy familiarizado, y funciona bien para mover un solo artículo. Piense en su correo electrónico de escritorio favorito cliente (Microsoft Outlook o Mail en OS X). ¿Cuándo arrastra un correo electrónico desde su bandeja de entrada y colóquelo en otra carpeta, que está haciendo una operación de movimiento. Lo mismo es cierto para arrastrar elementos a la Papelera de reciclaje (Windows) o Papelera (OS X), el elemento se mueve a el receptáculo.



Figure 22.2 Notice that the target now has two of the source's items.

TIP Usted todavía puede hacer una copia con `dragMoveEnabled` establecido en `true`, para ello, mantenga pulsada la tecla `Ctrl` (Comando en Mac) durante el proceso de arrastre. La inversa no es cierto cuando `dragMoveEnabled` es falso; manteniendo la tecla `Ctrl` o Comando no invoca un movimiento.

`dragMoveEnabled` permite pasar a los objetivos, pero lo que si desea ordenar una lista de elementos dentro de la misma lista? No se preocupe. Usando las tres propiedades de arrastre permite fácilmente lograr esto, como se muestra en la siguiente sección.

22.2.4 Uso de D & D para la clasificación controlada por el usuario

D & D no sólo se encarga de cómo mover o copiar datos de un componente a otro. También se puede utilizar para cambiar la posición de los datos en el mismo componente..

Este tipo de mecanismo de clasificación anticuada es muy usada en toda la web hoy en día.

El uso de D & D, usted puede ahorrar a usted ya sus usuarios una gran cantidad de tiempo.

Establezca las siguientes propiedades de un componente para permitir arrastrar y soltar desde y para el propio componente:

- `dragEnabled="true"`
- `dropEnabled="true"`
- `dragMoveEnabled="true"`

El ejemplo en el listado 22.2 muestra mediante el uso de una versión actualizada del listado 22.1 's

Enumere permite tanto arrastrar y soltar.

Listing 22.2 Dragging and dropping within a component, to allow ordering

```
<?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">
  <s:layout>
    <s:VerticalLayout paddingLeft="20" paddingTop="20" />
  </s:layout>
  <fx:Declarations>
    <s:ArrayCollection id="authors">
      <fx:Object label="Dan Orlando" telephone="555-0000" />
      <fx:Object label="Tariq Ahmed" telephone="555-0001" />
      <fx:Object label="John C. Bland II" telephone="555-0010" />
      <fx:Object label="Joel Hooks" telephone="555-0011" />
    </s:ArrayCollection>
  </fx:Declarations>
  <s:Label fontWeight="bold" text="My Favorite Authors" />
  <s>List dragEnabled="true" dropEnabled="true"
         dragMoveEnabled="true"
         dataProvider="{authors}" />
</s:Application>
```

Set drag properties accordingly

Después de ejecutar listado 22.2, se puede hacer algo similar a figura 22.3 y controlar el orden de sus autores favoritos.

Prometemos que la especie era puramente al azar, después de que es seleccionado el artículo, y nada está implícito en el orden de clasificación (* guiño guiño *).

My Favorite Authors



Figure 22.3 Using D&D on itself, the `List` lets the user easily control the position of items.

22.2.5 Multi-punto de arrastrar y soltar

Ya viste cómo mover y copiar objetos funciona bien cuando usted está tratando con sencillo listas de datos, pero ¿qué pasa cuando se quiere eliminar 10 artículos? 20 artículos? ¿Qué tal 5 elementos no consecutivos? Flex responde a la llamada, permitiendo varios artículos D & D. Si desea permitir al usuario mover muchos elementos a la vez, echa un vistazo a la `AllowMultiple-Propiedad Selection`. Si se establece en `true`, se activa esta capacidad (véase la siguiente lista).

Listing 22.3 Enabling the user to move many items at once

```
<?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">
  <s:layout>

    <s:HorizontalLayout paddingLeft="20" paddingTop="20" />
  </s:layout>
  <fx:Declarations>
    <s:ArrayCollection id="authors">
      <fx:Object label="Dan Orlando" telephone="555-0000" />
      <fx:Object label="Tariq Ahmed" telephone="555-0001" />
      <fx:Object label="John C. Bland II" telephone="555-0010" />
      <fx:Object label="Joel Hooks" telephone="555-0011" />
    </s:ArrayCollection>
  </fx:Declarations>
  <s:VGroup>
    <s:Label fontWeight="bold" text="Source" />
    <s>List dragEnabled="true" dragMoveEnabled="true"
           allowMultipleSelection="true"
           dataProvider="{authors}" />
  </s:VGroup>
  <s:VGroup>
    <s:Label fontWeight="bold" text="Target" />
    <s>List dropEnabled="true" dataProvider="{new ArrayCollection()}" />
  </s:VGroup>
</s:Application>
```

Set the
List's D&D
properties

El usuario puede seleccionar varios elementos en una de dos maneras: mediante selecciones individuales o usando una variedad de selecciones.

Para seleccionar los elementos uno por uno, siga los pasos indicados en la tabla 22.4.

Tipo

Pasos

Selecciones individuales

1 Mantenga pulsada la tecla Ctrl y utilizar el ratón para

seleccionar cada pieza que desea arrastrar.

2 Una vez que todos los elementos están seleccionados, suelte la tecla Ctrl.

Rango de selecciones

3 Haga clic en el primer elemento del rango.

4 Pulse y mantenga pulsada la tecla Shift.

5 Seleccione el último elemento del rango.

De cualquier manera que elija los resultados en un conjunto seleccionado de varios elementos. Una vez que tenga los elementos seleccionados, haga clic en cualquiera de los elementos seleccionados y arrastrarlo a la meta o, si que se desplazan dentro del mismo componente, a la posición de destino en la lista de datos.

Figura 22.4 Utilizando D & D para la clasificación controlada por el usuario

Este comportamiento de la interfaz de usuario no es exclusivo de Flex, y es algo es muy probable que haya experimentado en un entorno de aplicaciones Windows. Piense en su aplicación de correo electrónico de escritorio que hablamos antes. En el ejemplo anterior Hablaron acerca de cómo mover un elemento, pero los clientes de correo permiten que tome varios elementos (rango o no consecutivos) y moverlos a su antojo. Ahora tiene el poder de dar tal funcionalidad en sus aplicaciones Flex.

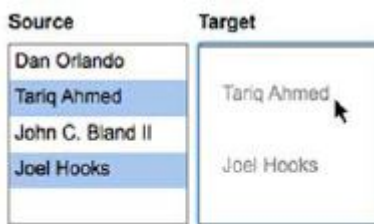


Figure 22.4 Use the Ctrl/Command key to select multiple items to drag.

Lo que hemos demostrado hasta ahora es un D & D operación unidireccional, donde hay una fuente y destino. Hay momentos en que usted necesita para mover un elemento de un componente a otro y luego de vuelta al otro componente. Se trata de dos vías de D & D, que Flex también apoya.

22.2.6 dos vías de arrastrar y soltar

A este punto, usted ha visto cómo mover o copiar elementos de una lista a otra. Este es grande, pero estamos hablando de D & D aquí, así que olvídate de un botón de deshacer, y vamos a permitir que al usuario arrastrar los artículos de nuevo al destino de origen. En Flex, eso no es un problema, y así es como lo haces.

Establezca las propiedades `dragEnabled` y `dropEnabled` a verdadera para todos los componentes involucrados. No olvides poner `dragMoveEnabled` de verdad también, si desea que el comportamiento default sea que se mueve en lugar de la copia, lo que hacemos en este caso.

Listado 22.4 muestra el mismo ejemplo de aplicación de los listados anteriores, pero con Ambas listas preparadas para D & D, que permite la comunicación de dos vías. Puede mover los elementos de ida y vuelta a través de las listas.

Listing 22.4 Two-way dragging and dropping between two List components

```
<?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">
  <s:layout>
    <s:HorizontalLayout paddingLeft="20" paddingTop="20" />
  </s:layout>
  <fx:Declarations>
    <s:ArrayCollection id="authors">
      <fx:Object label="Dan Orlando" telephone="555-0000" />
      <fx:Object label="Tariq Ahmed" telephone="555-0001" />
      <fx:Object label="John C. Bland II" telephone="555-0010" />
      <fx:Object label="Joel Hooks" telephone="555-0011" />
    </s:ArrayCollection>
  </fx:Declarations>
  <s:VGroup>
    <s:Label fontWeight="bold" text="Source" />
    <s:List dropEnabled="true" dragEnabled="true" dragMoveEnabled="true"
            allowMultipleSelection="true" dataProvider="{authors}" />
  </s:VGroup>
  <s:VGroup>
    <s:Label fontWeight="bold" text="Target" />
    <s:List dropEnabled="true" dragEnabled="true" dragMoveEnabled="true"
            allowMultipleSelection="true" dataProvider="(new ArrayCollection())" />
  </s:VGroup>
</s:Application>
```

Al mover un elemento de una lista a otra, se elimina de la ubicación de origen. Siempre se puede mover de nuevo el uso de D & D. Figura 22.5 muestra arrastrando desde la fuente Lista para la lista de destino y viceversa.



Figure 22.5 The left side shows dragging from source to target, and the right is vice versa.

Hasta ahora, usted ha sido capaz de sumar D & D de manera autónoma: Componentes emplean su función de D & D comportamiento. Si la vida fuera así de fácil! Con Flex 4 pelado se puede, la mayoría de las veces, se salen con la utilización de los componentes incorporados en y sin tener que escribir su propia lógica de D & D. En una aplicación real, tendrás que entrar en negocio lógicas y reglas de negocio para los componentes personalizados.

Adición de drag-and-drop para componentes personalizados está lejos de ser tan simple como una o más de tres propiedades, pero no está cerca de tirar los dientes, si eso ayuda a cualquiera. En el siguiente sección se explora completamente añadiendo D & D para los componentes personalizados.

DragManager

El DragManager es una clase de métodos estáticos, que hace todo el trabajo detrás de las escenas para hacer que el D & D sea posible.

DragManager propiedades y métodos

El DragManager tiene cuatro valores estáticos (también llamados constantes, que son variables cuyo valor no se puede cambiar en tiempo de ejecución) para poder evaluar la acción del usuario (s), tales como un movimiento o una copia, y comunicar al usuario visualmente.

Constantes definidas en el dragManager

COPIA – La acción actual es copiar el elemento arrastrado.

- Componentes basados en List asumen, cuando se está manteniendo pulsada la tecla Ctrl, estás haciendo una copia (si dragMoveEnabled = "true").

ENLACE- La acción actual es la vinculación, como una copia, pero en vez de hacer un duplicado, es tener varios elementos que unen a una sola fuente.

- Esta acción se produce cuando la tecla Shift esta presionada.

MOVER- La acción actual es mover el elemento arrastrado de un lugar a otro.

NINGUNO- No se permite la acción. La operación se negó.

Estos valores constantes se utilizan típicamente para establecer una comparación. El siguiente código muestra cómo saber si la acción actual es una copia:

```
protected function onDragEnter(event:DragEvent):void{
if(event.action == DragManager.COPY){
//do something
}
}
```

Usted puede tomar ventaja de algunos métodos importantes a fin de realizar personalizaciones D & D en su trabajo:

- doDrag () inicia el proceso de D & D. Por lo general, se inició en un evento mouseDown manejador.

- acceptDragDrop (): permite al DragManager saber si se trata de permitir que continúe el proceso de D & D.

- Llamado en un controlador de eventos dragEnter.

- ShowFeedback ()-Se comunica al usuario sobre el valor de la operación actual (COPIA, ENLACE, MOVE, o ninguno).

Usando una combinación de constantes del DragManager y sus métodos de apoyo, que tienen las piezas necesarias para poner en práctica su propia lógica.

Limitar el acceso.

El evento clave para facilitar la adición de un guardián de puerta lógica-negocio para determinar si permitir un drop, es el evento dragEnter. Este evento se activa cuando el ratón se mueve en un destino de colocación potencial.

Usemos un ejemplo, dónde podemos ver que en el destino permite la colocación sólo después de inspeccionar el objeto y verificar si el registro está activo para el arrastre.

Estos son los pasos a seguir en el código:

- 1 Crear un controlador de eventos dragEnter al listado para aceptar o rechazar un drop.
- 2 El controlador de eventos se ve en el evento DragEvent para ver lo que datos se están arrastrado, dichos datos están forma de un vector de objetos.
- 3 Examine los datos para ver si alguno de los artículos están activos (vía isActive =true or false).
- 4 Si todo está bien, dígame al DragManager para continuar, de lo contrario, volver atrás, y actualizar la información visual.

Ejemplo 22.5: Permite la caída únicamente a los objetos que son activos.

```
<?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">
<s:layout>
<s:HorizontalLayout paddingLeft="20" paddingTop="20" />
</s:layout>
<fx:Script>
<![CDATA[
import mx.core.UIComponent;
import mx.events.DragEvent;
import mx.managers.DragManager;
protected function onDragEnter(event:DragEvent):void{
var items:Vector.<Object> =
event.dragSource.dataForFormat("itemsByIndex")    ← | Acceso a los
                                                    datos arrastrados

as Vector.<Object>;
if(items[0].isActive)    ← | Verificar valor de propiedad
                        isActive**
DragManager.acceptDragDrop(event.target ← | Acepta el drop
as UIComponent);
else{
DragManager.showFeedback(DragManager.NONE);    ← actualiza la
informacion visual
event.preventDefault();    ← evento preventDefault *
}
}
}]>
</fx:Script>
<fx:Declarations>
<s:ArrayCollection id="authors">
<fx:Object label="Dan Orlando" telephone="555-0000"
isActive="false" />
<fx:Object label="Tariq Ahmed" telephone="555-0001"
isActive="false" />
<fx:Object label="John C. Bland II" telephone="555-0010"
isActive="true" />
<fx:Object label="Joel Hooks" telephone="555-0011"
isActive="true" />
</s:ArrayCollection>
</fx:Declarations>
<s:VGroup>
<s:Label fontWeight="bold" text="Source" />
<s>List dragEnabled="true" dragMoveEnabled="true"
dataProvider="{authors}" />
```

```

</s:VGroup>
<s:VGroup>
<s:Label fontWeight="bold" text="Target" />
<s:List dropEnabled="true" dataProvider="{new ArrayCollection()}"
dragEnter="onDragEnter(event)" /> ← se agrega el manejador del
evento dragEnter
</s:VGroup>
</s:Application>

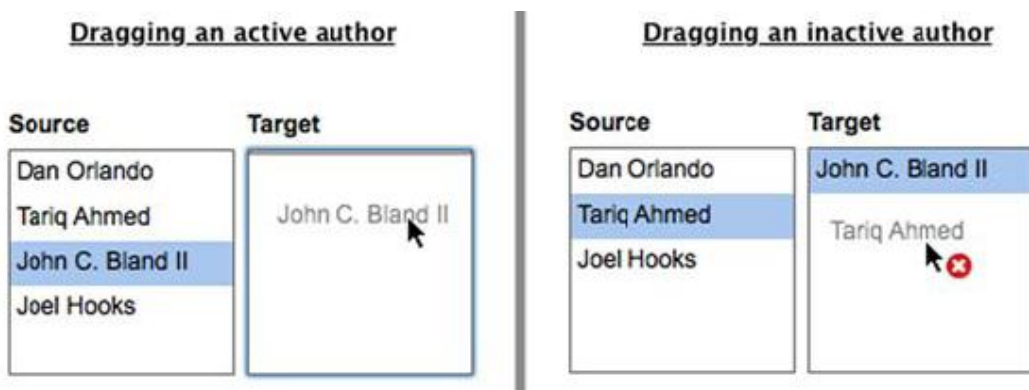
```

*El evento `preventDefault` lo que hace es cancelar el comportamiento predeterminado de un evento, si ese comportamiento puede ser cancelado.
 ** `isActive` es una propiedad personalizada del tipo de datos. No es parte del SDK de Flex 4 o originaria de una operación de D & D.

Al ejecutar este ejemplo, si arrastra el autor "Tariq Ahmed," la lista no permitirá que el drop se produzca. Una parte fundamental de este trabajo es la llamada al evento `event.preventDefault()`. Ya que si no se detiene el evento, el componente de lista, internamente aceptaría la caída. Al impedir el default, matas dicho comportamiento.

En la figura se muestran los resultados de arrastrar un autor activo e inactivo hacia el componente de destino.

22.6



Los componentes drag-and-drop utilizan diferentes formas de obtención de los elementos arrastrados y diferentes tipos. Para acceder a los elementos arrastrados desde una `mx:DataGrid`, tendrá que utilizar el siguiente código:

```
//halo
```

```
artículos var: Array = event.dragSource.dataForFormat ("items") como matriz;
```

```
//spark
```

```
elementos: var. Vector <Object> =
```

```
event.dragSource.dataForFormat ("itemsByIndex") como Vector <Object>;
```

La diferencia está en el formato disponible en el `dragSource`. Spark componentes no proporcionan un formato de artículos, mientras que los componentes de Halo sí.

Personalización del Drag & Drop

Es probable encontrarse con la necesidad de aplicar el dop por uno mismo. Usted podría eludir la funcionalidad por defecto, como lo hicimos en el listado 22.5 para la aceptación de un drop, para filtrar los elementos arrastrados, o es posible que desee almacenar la información en una base de datos al finalizar el dropped.

Para ello se le agrega al controlador de eventos `DragDrop` para tomar los datos en movimiento y copiarlo al `dataProvider` del componente. Porque estamos implementando nuestra propia funcionalidad movimiento, vamos a apagar `dragMoveEnabled` en el

componente fuente

Listado 22.6 Creación de una función para añadir datos directamente a un componente

```
<?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">
<s:layout>
<s:HorizontalLayout paddingLeft="20" paddingTop="20" />
</s:layout>
<fx:Script>
<![CDATA[
import mx.core.UIComponent;
import mx.events.DragEvent;
import mx.managers.DragManager;
protected function onDragDrop(event:DragEvent):void{
var target:List = event.target as List;
if(!target.dataProvider)
target.dataProvider = new ArrayCollection();
var items:Vector.<Object> =
event.dragSource.dataForFormat("itemsByIndex") as Vector.<Object>;
var sourceIndex:int;
var source:List = event.dragInitiator as List;
var sourceData:ArrayCollection = source.dataProvider
as ArrayCollection;
for each(var item:Object in items){
if(item.isActive){
target.dataProvider.addItem(item);
sourceIndex = source.dataProvider.getItemIndex(item);
source.dataProvider.removeItemAt(sourceIndex);
}
}
event.preventDefault();
}
}]>
</fx:Script>
<fx:Declarations>
<s:ArrayCollection id="authors">
<fx:Object label="Dan Orlando" telephone="555-0000"
isActive="false" />
<fx:Object label="Tariq Ahmed" telephone="555-0001"
isActive="false" />
<fx:Object label="John C. Bland II" telephone="555-0010"
isActive="true" />
<fx:Object label="Joel Hooks" telephone="555-0011"
isActive="true" />
</s:ArrayCollection>
</fx:Declarations>
<s:VGroup>
<s:Label fontWeight="bold" text="Source" />
<s:List dragEnabled="true" allowMultipleSelection="true"
dataProvider="{authors}" />
```

```

</s:VGroup>
<s:VGroup>
<s:Label fontWeight="bold" text="Target" />
<s:List dropEnabled="true" dragDrop="onDragDrop(event)" />
</s:VGroup>
</s:Application>

```

Listado 22.6 es similar a 22.5 pero proporciona mucho más control sobre el filtrado. Recuerde que en el 22,5 se le negó el conjunto de los elementos arrastrados si el primer elemento es inactivo.

Podrías haber recorrido sobre la lista y permitir la caída de si había un tema activo, pero esto habría permitido elementos inactivos también. Para evitar que esto ocurra apagamos `allowMultipleSelection` y obligándonos a arrastrar un solo elemento al mismo tiempo. Esto funcionó, pero no es lo ideal.

En el listado 22.6 permitimos selección múltiple, no se moleste con el evento `dragEnter`, y filtrar los elementos del controlador `dragdrop`. De esta manera estamos en condiciones de determinar cada estado del elemento (activo o inactivo) y luego decidir si queremos añadir el artículo o no. También establecemos `dataProvider` del destino si no está ya establecido. La figura 22.7 muestra arrastrando todos los elementos de origen y el resultado de la fricción.



Como se señaló anteriormente, es posible que desee guardar estos datos en la base de datos. Desde aquí se puede hacerlo en el controlador `dragdrop` mediante el almacenamiento de una lista de elementos y luego enviar sólo aquellos válidos, como se muestra en el siguiente fragmento:

```

...
var savedItems:ArrayCollection =
new ArrayCollection();           ← crea el lugar para guardar los
                                objetos
for each(var item:Object in items){
if(item.isActive){
target.dataProvider.addItem(item);
sourceIndex = source.dataProvider.getItemIndex(item);
source.dataProvider.removeItemAt(sourceIndex);
savedItems.addItem(item);       ← Guarda los objetos
aprobados
}
}
if(savedItems.length > 0){      ← verifica si hay objetos para
                                guardar

```

```
//insert database save logic here
objetos a la bd.
}
...
```

← Mandar a guardar los

El fragmento anterior es un pellizco a continuación del ejemplo 22.6. El único cambio es el uso del

ArrayCollection savedItems para almacenar una referencia a los artículos guardados. A continuación,

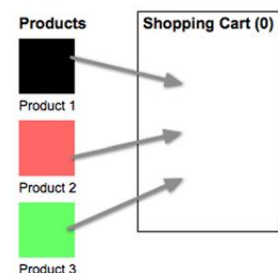
Verificamos si los artículos se almacenan para guardar, y si es así, le enviamos los artículos a nuestra base.

Adición de D & D a los componentes que no son Listas

22.8

Configuración del ejemplo

En este ejemplo se utiliza el concepto de un carro de compras (rudimentario), donde usted elige que comprar arrastrándolos y soltándolos en un área designada (véase la figura 22.8).



Iniciando el arrastre (Drag)

Nuestros productos están representados por una clase ProductItem, que se extiende SkinnableComponent, y encapsulan la funcionalidad arrastre (drag). No nos importa la funcionalidad soltar (drop) para los productos porque nada caerá en ellos, sino que solamente se arrastran.

Para iniciar el proceso de D & D, se le escucha al evento mouseMove y se crea un controlador de eventos al utilizar el DragManager. Esta función drag-initiator, realiza tres tareas principales:

- crear una referencia al drag-initiator
- Crea un objeto drag-source que contenga todos los datos que necesita para pasar hacia el DragManager
- Llamadas a funciones doDrag () del DragManager y pasa a lo largo de la referencia del Drag-initiator, el objeto drag-source y el objeto de evento mouseMove creados por el usuario.

Listado 22,7 ProductItem clase personalizada

```
package components{
import assets.graphics.DragProxyGraphic;
import flash.events.MouseEvent;
import mx.core.DragSource;
import mx.managers.DragManager;
import spark.components.Label;
import spark.components.supportClasses.SkinnableComponent;
import spark.primitives.supportClasses.FilledElement;
public class ProductItem extends SkinnableComponent{
[SkinPart(required="true")]
public var imageElement:FilledElement;
[SkinPart(required="true")]
public var titleElement:Label;
[Bindable]
public var title:String;
```

```

[Bindable]
public var color:uint;
public function ProductItem(){
super();
addEventListener(MouseEvent.CLICK, onMouseDown, false, 0, true);
}
protected function onMouseDown(event:MouseEvent):void{
var ds:DragSource = new DragSource();    ← 1)Crea la instancia
DragSource
ds.addData(this, "product");            ← 2)Agrega datos
personalizados al DragSource
DragManager.doDrag(this, ds, event);    ← 3)Comienza a
arrastrar
}}}

```

Para probar el arrastre, ha creado tres instancias de la clase ProductItem y un aspecto personalizado. El skin se muestra en el siguiente código:

```

<?xml version="1.0" encoding="utf-8"?>
<s:Skin xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark">
<fx:Metadata>[HostComponent("components.ProductItem")]</fx:Metadata>
<s:layout><s:VerticalLayout /></s:layout>
<s:Rect id="imageElement" width="50" height="50">
<s:fill><s:SolidColor color="{hostComponent.color}" /></s:fill>
</s:Rect>
<s:Label id="titleElement" text="{hostComponent.title}" />
</s:Skin>

```

El skin en su lugar, sólo se necesita una aplicación para probar. Listado 22.8 muestra nuestra aplicación inicial de las instancias ProductItem en su lugar.

```

<?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:components="components.*">
<s:layout>
<s:HorizontalLayout paddingLeft="20" paddingTop="20" />
</s:layout>
<fx:Script>
<![CDATA[
import assets.skins.ProductItemSkin;
]]>
</fx:Script>
<s:VGroup width="100">
<s:Label text="Products" fontWeight="bold" fontSize="14"
paddingTop="5"/>
<components:ProductItem title="Product 1"
color="0x333333"
skinClass="assets.skins.ProductItemSkin" />
<components:ProductItem title="Product 2"

```

```

color="0xFF6666"
skinClass="assets.skins.ProductItemSkin" />
<components:ProductItem title="Product 3"
color="0x66FF66"
skinClass="assets.skins.ProductItemSkin" />
</s:VGroup>
</s:Application>
*Se crean 3 instancias de ProductItem

```

La figura 22.9 muestra cómo la aplicación busca en su actual Estado mientras se arrastra del producto 1.

El uso de un drag-proxy personalizado

22.9



Por defecto Flex dibuja una caja simple para ilustrar un artículo que está arrastrando, llamado drag-proxy. Lo bueno es tener control sobre la apariencia de este proxy. Es simple de añadir un drag-proxy personalizado. El siguiente código muestra la personalización:

```

<?xml version="1.0" encoding="utf-8"?>
<s:Graphic xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark" >
<s:Path data="M 119,0 L 148,86 238,86 166,140 192,226 119,175
46,226
72,140 0,86 90,86 Z" y="2" scaleX="0.3361345" scaleY="0.3361345">
<s:fill><s:SolidColor color="#333333" /></s:fill>
<s:stroke><s:SolidColorStroke color="white" /></s:stroke>
</s:Path>
</s:Graphic>

```

Esto no es más que un componente MXML con un camino de elaboración de una estrella. Para utilizar este proxy personalizado, es necesario actualizar el método de la lista a DragManager.doDrag 22.8 y pasarle una instancia del proxy:

```
DragManager.doDrag(this, ds, event, new DragProxyGraphic());
```

Para un verdadero carro del envío usted probablemente querrá usar gráfica del producto como el drag-proxy. El siguiente fragmento de código muestra cómo se puede pasar una imagen para representar con precisión cuál es el producto en ser arrastrado:

```

protected function onMouseDown(event:MouseEvent):void{
var screenshot:BitmapData = new BitmapData(width, height);
screenshot.draw(this);
var proxy:Image = new Image();
proxy.source = new Bitmap(screenshot.clone());
var ds:DragSource = new DragSource();
ds.addData(this, "product");
DragManager.doDrag(this, ds, event, proxy);
}

```

Este ejemplo muestra como "imagen" del componente de arrastre y el uso de dicha imagen como el drag-proxy. La figura 22.11 muestra los resultados del fragmento anterior.

Products



Figure 22.10 Using a custom drag proxy instead of the default box graphic

Products



Figure 22.11 Using a bitmap screenshot of the dragging component

Manejo del drop

U va a construir otro componente personalizado.

En este componente, usted va a buscar los eventos `dragEnter`, `dragExit` y `DragDrop`.

Usted también va a realizar el seguimiento del número total de elementos que añade a la cesta de la compra y luego utilizar los estados para actualizar visualmente la pantalla y mostrar que los artículos están en el carro

Listado 22.9 La clase `ShoppingCart` se utiliza para manejar el dropping

```
package components{
import flash.events.Event;
import mx.events.DragEvent;
import mx.managers.DragManager;
import spark.components.Group;
import spark.components.Label;
import spark.components.supportClasses.SkinnableComponent;
import spark.filters.BlurFilter;
[SkinState("noitems")]
[SkinState("items")]
public class ShoppingCart extends SkinnableComponent{
[SkinPart(required="true")]
public var titleElement:Label;
[SkinPart(required="true")]
public var contentGroup:Group;
[Bindable]
public var title:String;
protected var _itemCount:uint = 0;
[Bindable(event="itemCountchanged")]
public function get itemCount():uint{ return _itemCount; }
public function ShoppingCart(){
super();
addEventListener(DragEvent.DRAG_ENTER,
onDragEnter, false, 0, true);
addEventListener(DragEvent.DRAG_DROP,
onDragDrop, false, 0, true);
addEventListener(DragEvent.DRAG_EXIT,
onDragExit, false, 0, true);
}
protected function onDragEnter(event:DragEvent):void{
if(event.dragSource.dataForFormat("product")    ←1)Verifica si el
dragSource
```

```

is ProductItem){
DragManager.acceptDragDrop(this);
filters = [new BlurFilter()];
visual(Blur)
}
}
protected function onDragExit(event:DragEvent):void{
filters = [];
filtros visuales
}
protected function onDragDrop(event:DragEvent):void{
var element:ProductItem =
event.dragSource.dataForFormat("product")
contentGroup.addElement(element);
para mostrar
_itemCount = contentGro.numElements;
dispatchEvent(new Event("itemCountChanged"));
invalidateSkinState();
filters = [];
}
override protected function getCurrentSkinState():String{
return _itemCount == 0 ? "noitems" : "items";
}}}

```

tiene un ProductItem
 ←2) Acepta el drop
 ←3) Agrega filtro
 ←4) Elimina todos los
 ←5) obtiene los
 ←6) Agrega elemento
 ←7) Actualizar conteo

Cuando un elemento se arrastra sobre este componente, compruebe para asegurarse de que los datos están en el formato adecuado **(1)**, y si lo es, aceptar el drop **(2)** y a continuación, añadir un filtro visual **(3)**. El filtro visual (blur) es una forma de ilustrar al usuario de que puede producirse un drop, como se muestra en figura 22.12.

Posteriormente, cuando el mouse usuario sale de la zona, se elimina el efecto borroso (blur) **(4)**.

Por supuesto, un efecto borroso no es probablemente la mejor manera de ilustrar la aceptación, pero funciona para con fines de demostración.

Lo que queda del proceso de D & D es tomar el elemento caído y agregarlo a la pantalla. Esto se hace en el controlador de eventos OnDragDrop. Debido a que la clase ProductItem se agrega como el dragged de datos **(5)**, usted toma los datos arrastrados y lo agrega a su lista contentGroup **(6)**. Debido a que el número de artículos en el carro ha cambiado, actualizar la propiedad _itemsCount con el número total de elementos en el contentGroup **(7)**.



La operación drop se ha completado, pero el componente todavía está borroso (efecto

blur), por lo que se eliminan todos los filtros.

Personalización de la experiencia de arrastrar y soltar

Antes de concluir este capítulo, nos desplazamos al aspecto creativo de D & D. Esto significa que usted puede cambiar sus propios iconos para los distintos gestos del usuario en la interfaz usuario. Ya viste cómo cambiar el drag-proxy.

Cambiar los iconos del drag-proxy

Como mencionamos anteriormente, el drag-proxy es la retroalimentación visual comunicado al usuario durante el proceso de D & D. La forma más fácil de cambiar es usar CSS / Estilos.

Desde la perspectiva del código, el siguiente código muestra cómo el uso de CSS puede definir qué imágenes son utilizadas por el drag-proxy para cada escenario de D & D más cómo configurar globalmente el propio drag-proxy:

```
<fx:Style>
@namespace mx "library://ns.adobe.com/flex/mx";
mx|DragManager{
copy-cursor: Embed(source="/assets/images/copy.png");
link-cursor: Embed(source="/assets/images/link.png");
move-cursor: Embed(source="/assets/images/move.png");
reject-cursor: Embed(source="/assets/images/reject.png");
default-drag-image-skin:
ClassReference("assets.skins.DefaultDragImageSkin");
}
</fx:Style>
```

Esta es, obviamente, mucho más fácil que cambiar la imagen de arrastre, y le permite aprovechar sus habilidades de CSS. Tenga en cuenta que el default-drag-image-skin sólo se utiliza cuando el drag-proxy no se establece explícitamente.

Mascaras para componentes List para arrastrar y soltar

Las siguientes secciones pequeñas muestran formas sencillas de personalizar tanto el arrastre del elemento y el indicador de drop.

CONTROLAR EL ELEMENTO DE ARRASTRE (DRAG)

El elemento de arrastre normalmente es un simple campo de etiqueta con el mismo color que la lista en sí.

Vamos a utilizar un elemento personalizado para cambiar el color de la fuente de arrastre a naranja, como se muestra en el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<s:ItemRenderer xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx"
autoDrawBackground="true" minHeight="15">
<s:states>
<s:State name="normal"/>
<s:State name="dragging"/>
```

```

</s:states>
<s:Label id="labelDisplay" fontWeight.dragging="bold"
color.dragging="0xde7800" />
</s:ItemRenderer>

```

Para utilizar este procesador, se hace referencia a él en la fuente de la lista de esta manera:

```

<s:List dragEnabled="true" dataProvider="{authors}"
itemRenderer="assets.renderers.ListItemRenderer" />

```

Este se encarga de la lista de origen, pero también queremos que modifique la lista de objetivos.

CONTROL DEL INDICADOR DEL DROP

Un indicador de drop muestra la ubicación del elemento arrastrado una vez que se dejó caer. Es un precursor para el drop y es puramente para una referencia visual para el usuario. Crear una copia de la Lista de la máscara por defecto y reemplaza el componente dropIndicator con el siguiente fragmento:

```

<fx:Component id="dropIndicator">
<s:Group minWidth="3" minHeight="3">
<s:Rect left="0" right="0" top="0" bottom="0">
<s:fill><s:SolidColor color="0xde7800" /></s:fill>
<s:stroke>
<s:SolidColorStroke color="0xde7800" weight="5"/>
</s:stroke>
</s:Rect>
</s:Group>
</fx:Component>

```

Para utilizar la nueva máscara, establezca la propiedad skinClass en la lista de destino:

```

<s:List dropEnabled="true" dataProvider="{new ArrayCollection()}"
skinClass="assets.skins.ListSkin"/>

```

Mezcla de arrastrar y soltar entre Spark y Halo

En el momento de esta publicación, no todos los componentes de Halo tienen equivalentes Spark. Este medio, como se ha mencionado en capítulos anteriores, que si desea utilizar una cuadrícula de datos que serán utilizando el Halo DataGrid. Eso está perfectamente bien, por ahora. La preocupación es si puede utilizar arrastrar y soltar entre los dos, y la respuesta es ¡sí!

Listado 22.12 muestra cómo funciona la D & D entre un DataGrid de Halo y un componente lista Spark

```

<?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">
<s:layout>
<s:HorizontalLayout paddingLeft="20" paddingTop="20" />
</s:layout>

```

```

<fx:Declarations>
<s:ArrayCollection id="authors">
<fx:Object label="Dan Orlando" telephone="555-0000" />
<fx:Object label="Tariq Ahmed" telephone="555-0001" />
<fx:Object label="John C. Bland II" telephone="555-0010" />
<fx:Object label="Joel Hooks" telephone="555-0011" />
</s:ArrayCollection>
</fx:Declarations>
<s:VGroup>
<s:Label fontWeight="bold" text="Source" />
<mx:DataGrid dragEnabled="true" dragMoveEnabled="true"
dataProvider="{authors}" />
</s:VGroup>
<s:VGroup>
<s:Label fontWeight="bold" text="Target" />
<s:List dropEnabled="true" dataProvider="{new ArrayCollection()}"
/>
</s:VGroup>
</s:Application>

```

Lo único que cambió de lista 22.1 es que el componente fuente está en un mx: DataGrid lugar de una: Lista.

No hay temas o peculiaridades cuando se datos de un componente halo de arrastrar y un componente de arrastrar y soltar Spark.



en

mueve
soltar a

Resumen

Arrastrar y soltar es interesante porque puede ser tan complicado como usted quiere que sea. Para la mayor parte, Flex hace todo el trabajo duro por usted, sólo tiene que interceptar los puntos en que desea interponer lógica personalizada.

Componentes basados en List vienen con soporte incorporado para hacer la mayor parte del trabajo, pero puede atrapar ninguno, algunos, o todos los eventos de D & D. Para componentes no basados en List, aprovechar el DragManager como su caballo de batalla. Flex requiere los controladores de eventos de arrastrar relacionados y pasa el objeto DragEvent cada vez, que le da la información con la que necesita trabajar.

La usabilidad es la razón principal por la que se está utilizando D & D, y aprovechar su capacidad para ahorrar tiempo a los usuarios.