

Personalizando la experiencia.

Integrantes del grupo :

Diego Martínez

Gonzalo Silva

Nicolás Castro

Horacio García

Personalizando la experiencia.

Nos vamos a enfocar en :

- * Diseño de la experiencia del usuario
- * El Modelo VIBE
- * Temas y pieles o máscaras
- * CSS en Flex 4
- * Mejores prácticas
- * Extensibilidad

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.

Construyendo alrededor de las historias del usuario.

Relacionadas con el concepto de escenarios de casos de uso.

Es construida alrededor de un personaje ficticio.

Consideración del contexto.

Categorías contextuales en las cuales se basa :

- * Diseño visual e interacción
- * La retroalimentación del usuario
- * Repuesta y Performance

Diseño visual e interacción.

Consiste en más que elementos visuales. Incluye navegación, interacción y referencias metafóricas.



Set de standard íconos que usan referencias metafóricas

Diseño visual e interacción.



Retroalimentación del usuario.

Se relaciona con el público para el cual se construye la aplicación. Mensajes inútiles de error harán que el usuario se sienta frustrado.

Respuesta y Performance.

En Flex no es raro llegar a un “deadlock”.
Se puede necesitar la creatividad para superar los obstáculos de rendimiento o performance.
La solución es adoptar mejores prácticas para acercarse a Flex.
El autor creó el modelo VIBE como una manera de simplificar el proceso.

El modelo VIBE

Visual appeal (Atractivo visual),
Interactive experience (Experiencia **I**nteractiva)
Bussines Optimazition (Optimización del
Negocio)

Extensibility (Extensibilidad)

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

La respuesta siempre debería limitarse a frases
cortas que sólo describen sentimientos y
emociones.

El modelo VIBE.

El modelo VIBE se usa para 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)

Visual appeal.

El diseño visual de una aplicación Flex es creado con el uso de temas, skins, 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 en Flex 4, llamado Spark. En Flex 3 el tema por defecto se llama Halo.

Aplicación de un tema.

Un tema se puede aplica mediante el argumento compilador de temas, por ejemplo utilizando Flash Builder

En cambio si no utiliza algún software se puede usar el compilador de línea de comandos.

Creando un tema SWC.

Un archivo de tema SWC contiene todas las subpartes que pertenecen al tema. El mismo incluye :

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

Creando un tema SWC.

Utilizar un archivo de configuración XML para compilar su tema SWC sería la forma más sencilla, el mismo especifica todos los archivos que usted quiere incluir en el SWC

Ejemplo : archivo XML de configuración.

<output>MyTheme.swc</output>

Define la salida arch. SWC resultado

<include-file>

Para CSS y archivos media

<include-classes>

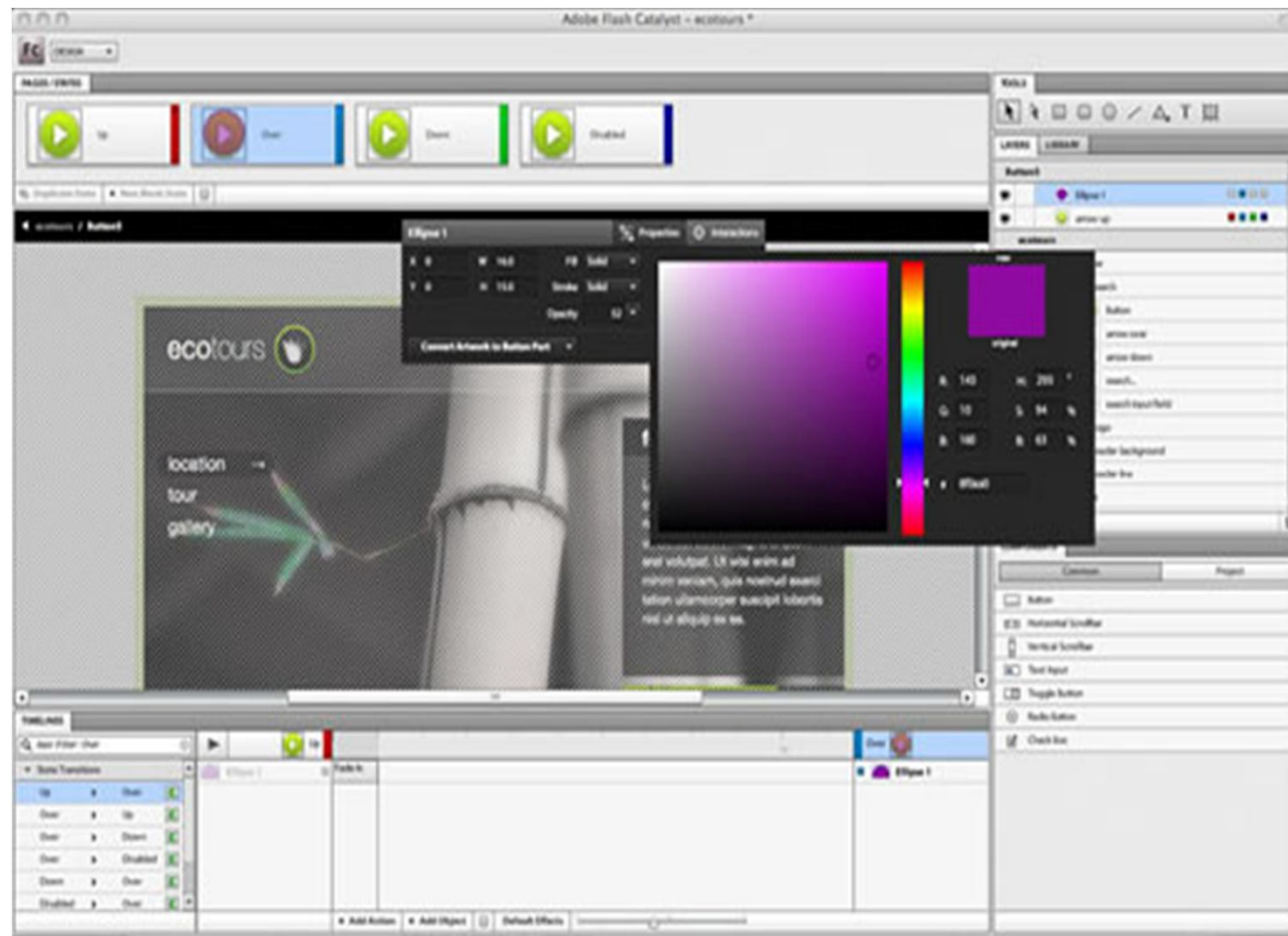
Para capas programadas

Dando estilo a las aplicaciones Flex 4 con CSS

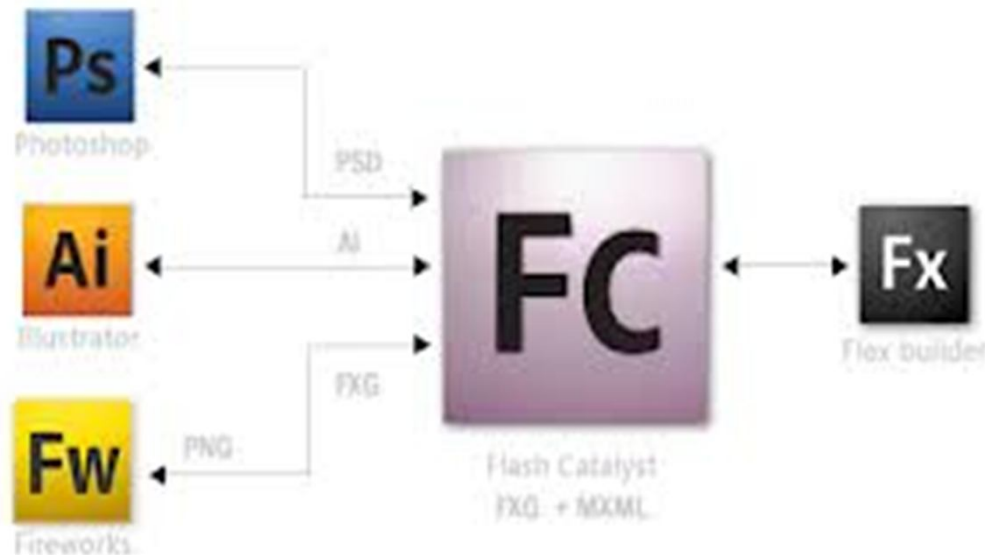
Las mejoras de CSS incluyen:

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

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 progr



Flash Catayst

- Puede abrir FXG archivos exportados desde cualquiera de los programas de diseño CS5
- Tiene la capacidad de exportar directamente un solo archivo FXP (Proyecto de Flex), que puede entonces ser perfectamente importado en Flash Builder.
- La capa de negocio de la aplicación, o la lógica de comportamiento, se agrega al proyecto de Flash Builder

Flujo de desarrollo

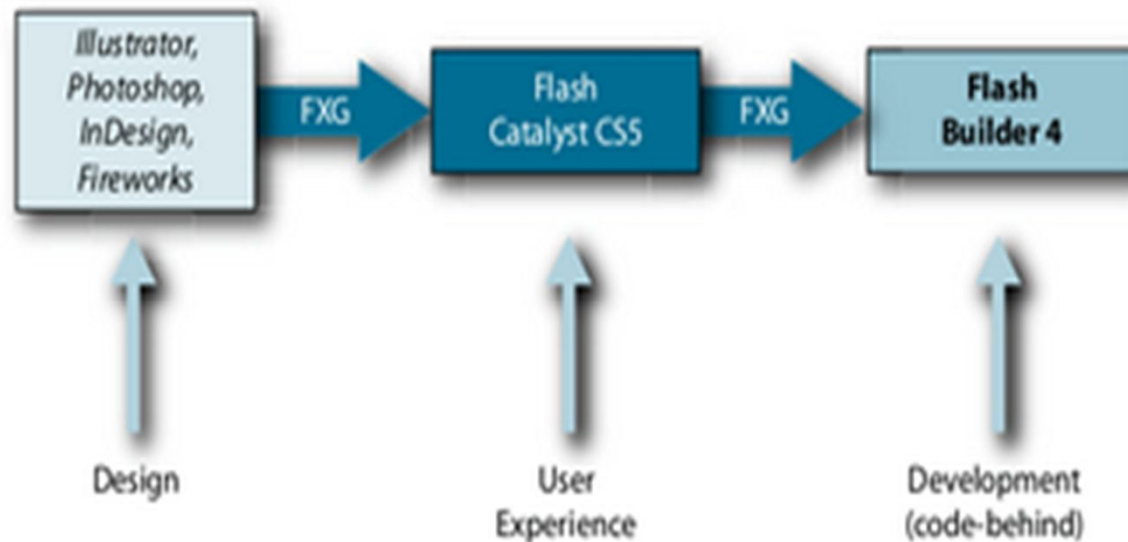


Figure 20.5 Moving from left to right, notice that FXG is used to translate graphics between the Adobe applications.

Mejorar la experiencia con los 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.
- Ejemplos: <http://blog.flexexamples.com/>

Efecto: Rotación 3D en el eje X

```
<?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"

        />
    </fx:Declarations>
    <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" />
</s:Application>
```

Place effects inside
Declarations tag

1 Declare values
and bindings

1 Declare values
and bindings

2 Effect triggered
using play()

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.

Efectos disponibles

| 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 |

Clases de efectos principales

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

Causa → 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 y accionar 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 un efecto. Vamos a explorar las diferentes disparadores usados para integrar efectos

Disparadores de Efectos

Evento: 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.

Disparado por Evento

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 de apoyo una serie de propiedades, lo que se usa para especificar los efectos que desea reproducir de forma automática en el evento correspondiente.

Disparado por Evento 2

| | |
|------------------------|---|
| addedEffect | El componente que se añade a un recipiente |
| creationCompleteEffect | El componente fue creado |
| focusInEffect | El teclado se centra en el componente |
| focusOutEffect el | El teclado centrándose fuera 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 |

Disparo Mediante Programación

Suponga que tiene una función de recordatorio, y si algo se retrasa desea hacer el recordatorio.

Este efecto se puede realizar contando los segundos, si se sobrepasa un umbral, un mensaje se ilumina en varias ocasiones

Disparo Mediante Transacción

La última manera de accionar un efecto, es a través de una transición de un estado a otro.

Se pueden definir transiciones deseadas para cada estado de un objeto.

También se pueden definir transacciones específicas para cambios de estado específicas

Drag & Drop

Drag & Drop = Arrastar y soltar

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 la 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 cualquier comportamiento predeterminado en todas las etapas del proceso.

El proceso D & D

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

Eventos de D & D

Drag Initiator: Si inicia el arrastre del componente.

Drag 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.

Eventos D & D 2

Cuál está en juego en ciertos momentos puede ser confuso, así que vamos a ir a través del proceso 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.

Pasos D & D

1 Se inicia Drag.

El usuario hace clic en un elemento 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.

Pasos D & D 2

2 Se crea Arrastre del objeto de origen.

Contiene todos los datos asociados con el elemento que se está arrastrando.

Por ejemplo, si usted está arrastrando una fila de un componente List, puede mostrar sólo la etiqueta, aunque la fila incluye 10 campos. Cuando se arrastra, parece que está arrastrando solo la etiqueta y en verdad esta arrastrando la totalidad de la fila.

Pasos D & D 3

3 se muestra una representación inicial arrastre visual.

Un icono rojo (x) indica que el componente del ratón está sobre un objeto que no acepta ser arrastrado.

A (+) icono verde se utiliza cuando se permite el arrastre.

Pasos D y D 4

4 Un drag representante destino se muestra como los ratones de usuario a través de los componentes.

Un rojo (x) icono indica que el ratón está sobre un componente que no se acepta la caída actual.

A (+) icono verde se utiliza cuando se permite que la caída actual.

Los sucesos activados: dragEnter y dragOver.

El destino es un destino de colocación y el destino componente examina el origen de arrastre para ver si le gusta lo que ve (Son los datos de tipo compatible? ¿Son ciertas reglas de negocio en su lugar? y así sucesivamente).

Pasos D & D 5

5 puntero del mouse del usuario en posible destino de colocación.

El proxy de arrastre se actualiza.

Evento disparó: dragExit.

Pasos D & D 6

6 usuario deja caer el arrastre en un objetivo.

Drag se ha completado o cancelado.

Si el componente aceptó se ha completado el D & D.

Si el componente rechazó, el arrastre se cancela y el artículo se devuelve a la fuente.

Los sucesos activados: dragdrop y dragComplete.

Eventos D & D

| Evento | Receptor |
|--------------|---|
| mouseDown | Se produce cuando el usuario hace click en el elemento arrastrable. |
| mouseMove | El Usuario comienza a mover el elemento arrastrable. |
| dragstart | Iniciador de arrastre dispara un evento en sí mismo. |
| dragEnter | 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. |
| dragOver | Se produce inmediatamente después de dragEnter y se envía sólo si el destino acepta el arrastre. Se puede utilizar para actualizar visualmente puntero del ratón demostrando que está listo para su aceptación. |
| dragdrop | Resultado de que el usuario suelta el botón del ratón. Aquí es donde se pasan los datos y finaliza la operación. |
| dragExit | Se produce si el usuario no suelte el botón del ratón y se aleja del destino de colocación, que van desde dragOver a dragExit. |
| dragComplete | Enviado al iniciador de arrastre cuando el proceso de D & D |

Componentes con soporte nativo D & D

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:

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

Habilitación de D & D en las listas

`dragEnabled`: Indica si el componente se le permite servir como un iniciador. verdadero o falso (por defecto).

`dropEnabled`: Indica si el componente se le permite servir como destino. verdadero o falso (por defecto).

Copiar - Pegar

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.

Copiar – Pegar 2

Los componentes basados en List admiten una propiedad `dragMoveEnabled`, que por defecto está establecida en `false` y hace que el comportamiento por defecto sea la copia de de datos..

Todo lo que necesitas hacer es establecer `dragMoveEnabled` en `true` y el D & D realiza un movimiento en lugar de una copia.

Otras Características

D & D también permite modificar el orden de una lista sin necesidad de interactuar entre mas de una.

También es posible realizar el movimiento múltiple de datos entre listas.

DragManager

- Es una clase de métodos estáticos, que hace todo el trabajo detrás de escena para hacer que el D & D sea posible.

DragManager propiedades y métodos

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  
    }  
}
```

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`.
- ❑ `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).

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.

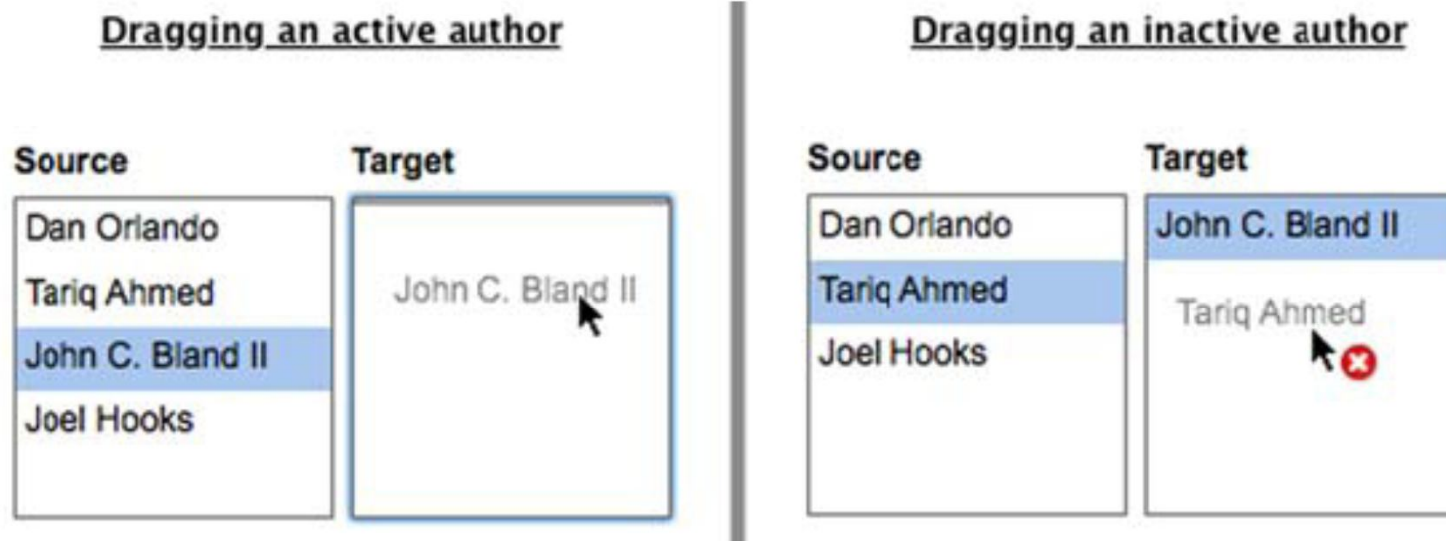
Usemos un ejemplo, dónde podemos ver que en el destino permite la colocación sólo después de inspeccionar el objeto y verificar si el registro está activo para el arrastre.

```

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 *
}
.....
<s:List dropEnabled="true" dataProvider="{new ArrayCollection()}"
dragEnter="onDragEnter(event)" /> ← se agrega el manejador del evento

```

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



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

Iniciando el arrastre (Drag)

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.

Ejemplo:

```
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  
}}
```

El uso de un drag-proxy personalizado

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>
```

```

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



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



Figure 22.11 Using a bitmap screenshot of the dragging component

Manejo del drop

Se va a construir otro componente personalizado.

En este componente, se va a buscar los eventos dragEnter, dragExit y DragDrop. Donde también va a realizar el seguimiento del número total de elementos que añade al contenedor, luego utilizar los estados para actualizar visualmente la pantalla y mostrar que los artículos están en el carro

```

protected function onDragEnter(event:DragEvent):void{
if(event.dragSource.dataForFormat("product")  ←1)Verifica si el dragSource
is ProductItem){                                tiene un ProductItem
DragManager.acceptDragDrop(this);              ←2)Acepta el drop
filters = [new BlurFilter()];                  ←3) Agrega filtro visual(Blur)
}
}
protected function onDragExit(event:DragEvent):void{
filters = [];                                  ←4)Elimina todos los filtros visuales
}
protected function onDragDrop(event:DragEvent):void{
var element:ProductItem =
event.dragSource.dataForFormat("product")      ← 5)obtiene los datos del drag
as ProductItem;
contentGroup.addElement(element);              ←6)Agrega elemento para mostrar
_itemCount = contentGro.numElements;          ←7) Actualizar conteo de items
dispatchEvent(new Event("itemCountChanged"));
invalidateSkinState();
filters = [];
}
override protected function getCurrentSkinState():String{
return _itemCount == 0 ? "noitems" : "items";
}}}

```

Initial view

Products



Product 1



Product 2



Product 3

Shopping Cart



Within onDragOver handler

Products



Product 1



Product 2



Product 3

Shopping Cart



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>
```