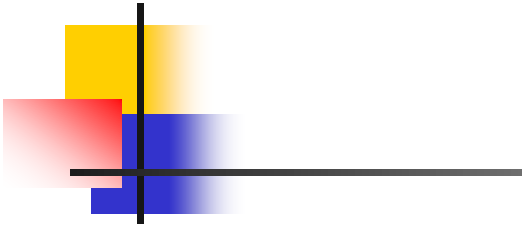


Flex 4.5

Le dynamisme applicatif
avec les états, les transitions
et les effets

Dynamisme, Transitions, effets

- Objectif du cours
  - Dans le chapitre précédent, nous avons vu comment agencer les composants au sein d'une interface
  - Nous allons à présent aborder les notions d'états, de transitions et d'effets permettant d'ajouter dynamisme et convivialité à nos applications Flex

Dynamisme, Transitions, effets

- Les états ou view states
  - En Flex, une interface graphique est composée d'un fichier MXML contenant des composants disposés suivant un agencement précis
  - Pour animer et donner vie à cette interface, on utilise la notion d'états :
  - Ainsi :
    - L'interface initiale donne lieu à un état de base
    - Cet état peut donner naissance à des états enfants ayant les mêmes caractéristiques que leur parent, mais pour lesquels il est possible d'ajouter ou de supprimer des composants ou encore de modifier les comportements et événements qui leur sont associés

Dynamisme, Transitions, effets

- Les états ou view states
  - Pour introduire la notion d'état, nous pouvons l'assimiler au changement de page d'un site web
  - Un état correspond alors à une nouvelle page web

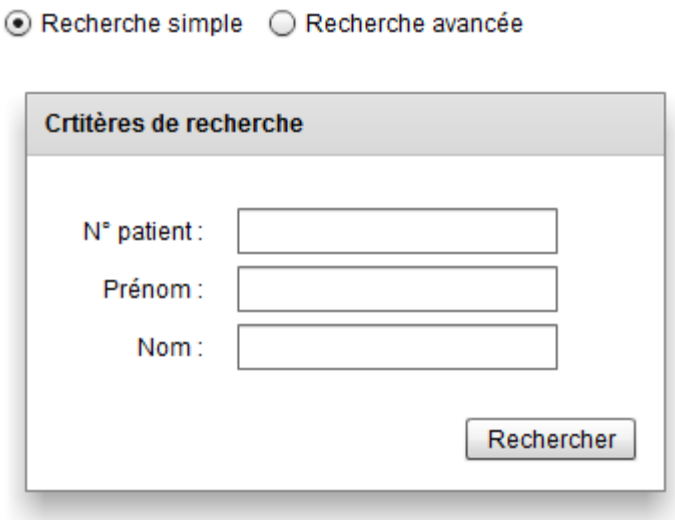
Dynamisme, Transitions, effets

Les états ou view states

- Exemple : création de l'état de base : BasicState.mxml
  - Imaginons une interface graphique permettant d'effectuer des recherches sur des patients dans une base de données médicales
  - Deux types de recherche sont possibles :
    - La recherche simple contenant quelques critères de base (numéro de patient, prénom et nom)
    - La recherche avancée comportant davantage de critères (numéro de chambre, date d'admission…)
    - La recherche basique sera proposée par défaut à l'utilisateur, mais il lui sera possible de basculer vers la recherche avancée en sélectionnant cette option

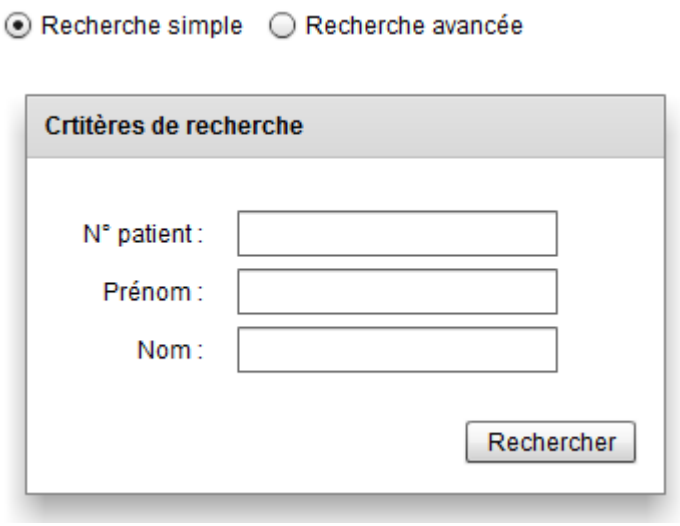
- Exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application …>
<fx:Declarations>
    <s:RadioButtonGroup id="radioBoutons"/>
</fx:Declarations>
<!-- GROUPE DE BOUTONS RADIO -->
<s:RadioButton x="34" y="21" label="Recherche simple"
    groupName="radioBoutons" id="btn_simple" selected="true"/>
<s:RadioButton x="166" y="21" label="Recherche avancée"
    groupName="radioBoutons" id="btn_avance"/>
```



- Exemple (suite):

```
<!-- CRITÈRES DE RECHERCHE -->
<s:Panel x="44" y="65" width="320" height="200" title="Crtitères de
    recherche">
<mx:Form x="10" y="10">
    <mx:FormItem label="N° patient :">
      <mx:TextInput id="numPatient"/>
    </mx:FormItem>
    <mx:FormItem label="Prénom :">
      <mx:TextInput id="Prenom"/>
    </mx:FormItem>
    <mx:FormItem label="Nom :">
      <mx:TextInput id="Nom"/>
    </mx:FormItem>
</mx:Form>
<s:Button x="219" y="131" label="Rechercher" id="btn_rechercher"/>
</s:Panel>
</s:Application>
```



Recherche avancée : twoStates.mxml

- Nous allons donc créer un état enfant qui héritera de l'état de base auquel nous ajouterons d'autres critères
- Pour créer ce nouvel état, nous allons ajouter le code suivant dans le fichier MXML de notre application :

```
<s:states>
    <s:State name="RechercheSimple"/>
    <s:State name="RechercheAvancee"/>
</s:states>
```

- Ajoutons-lui à présent les critères de recherche supplémentaires que sont :
  - la chambre
  - et la date d'admission

Exemple : twoStates.mxml

```
<?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" minWidth="955"
     minHeight="600">
<s:states>
    <s:State name="RechercheSimple"/>
    <s:State name="RechercheAvancee"/>
</s:states>
<fx:Declarations>
    <s:RadioButtonGroup id="radioBoutons"/>
</fx:Declarations>
<!-- GROUPE DE BOUTONS RADIO -->
<s:RadioButton x="34" y="21" label="Recherche simple"
    groupName="radioBoutons" id="btn_simple" selected="true"/>
<s:RadioButton x="166" y="21" label="Recherche avancée"
    groupName="radioBoutons" id="btn_avance"/>
```

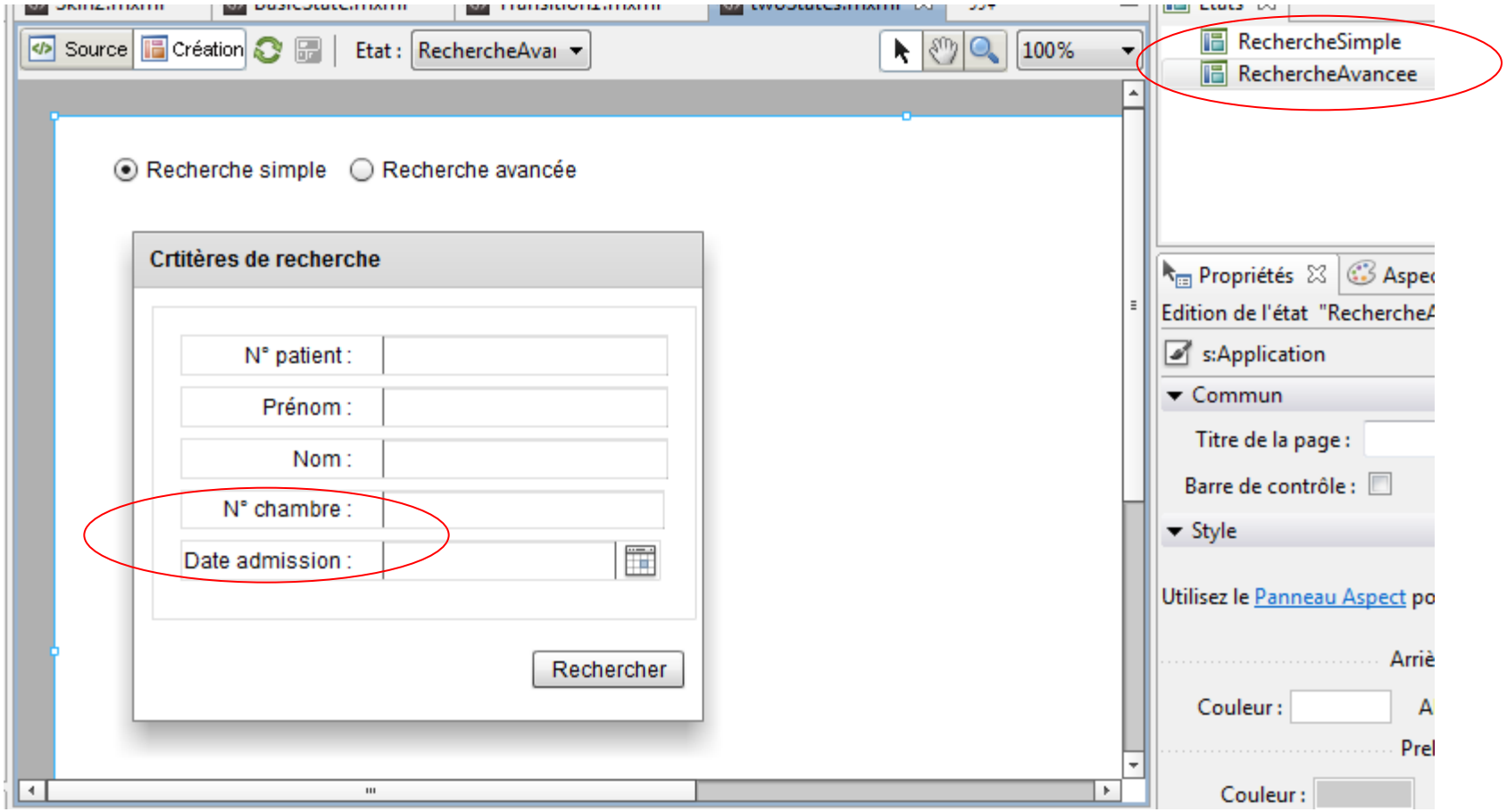
```
<!-- CRITÈRES DE RECHERCHE -->
<s:Panel x="44" y="65" width="320" height="200" title="Crtitères de
    recherche" height.RechercheAvancee="275">
    <mx:Form x="10" y="10" height.RechercheAvancee="176">
      <mx:FormItem label="N° patient :">
                <mx:TextInput id="numPatient"/>
      </mx:FormItem>
      <mx:FormItem label="Prénom :">
                <mx:TextInput id="Prenom"/>
      </mx:FormItem>
      <mx:FormItem label="Nom :">
                <mx:TextInput id="Nom"/>
      </mx:FormItem>
      <mx:FormItem includeIn="RechercheAvancee" label="N° chambre :">
                <s:TextInput width="158"/>
      </mx:FormItem>
      <mx:FormItem includeIn="RechercheAvancee" label="Date admission :">
                <mx:DateField width="156"/>
      </mx:FormItem>
    </mx:Form>
    <s:Button x="219" y="131" label="Rechercher" id="btn_rechercher"
    x.RechercheAvancee="223" y.RechercheAvancee="203"/>
</s:Panel>
```

Remarques

1. **Pour spécifier** des caractéristiques supplémentaires aux composants situés dans un état enfant, il suffit de suffixer les attributs de ces composants par le nom de l'état enfant
2. **Pour ajouter** de nouveaux composants, les inclure par **includeIn**

Pour visualiser le résultat obtenu

- Placez-vous dans la perspective Design de Flash Builder
- En haut à droite de cette perspective, vous trouverez la vue States
- Si le code a été correctement saisi, vous devriez visualiser le nouvel état
- Vous pouvez également naviguer entre les différents états en cliquant dessus, ce qui permet d'en visualiser les différences


Source
Création
Etat :
RechercheAvai
100%
RechercheSimple
RechercheAvancee
Recherche simple
Recherche avancée
Crtitères de recherche
N° patient :
Prénom :
Nom :
N° chambre :
Date admission :
Rechercher
Propriétés
Edition de l'état "RechercheA
s:Application
Commun
Titre de la page :
Barre de contrôle :
Style
Utilisez le Panneau Aspect po
Couleur :
Couleur :

Dynamisme, Transitions, effets

Les états ou view states

- Mise en place de l'interactivité
  - Le passage d'un état à un autre sera effectué lors de la sélection du type de recherche à l'aide des boutons radio
  - C'est donc sur l'événement click de ceux-ci que nous ferons appel à l'état correspondant à l'option sélectionnée
  - La sélection d'un état s'effectue grâce à la méthode currentState à laquelle nous passons en paramètre l'état à afficher :

Dynamisme, Transitions, effets

Les états ou view states

- Mise en place de l'interactivité (suite)

```
<!-- GROUPE DE BOUTONS RADIO -->
<s:RadioButton x="34" y="21" label="Recherche simple"
    groupName="radioBoutons" id="btn_simple"
    selected="true" click="currentState=''"/>
<s:RadioButton x="166" y="21" label="Recherche avancée"
    groupName="radioBoutons" id="btn_avance"
    click="currentState='RechercheAvancee'"/>
```

Dynamisme, Transitions, effets

Les états ou view states

- Les transitions
  - Un peu à la manière d'un logiciel de montage vidéo, Flex permet d'effectuer des transitions entre chaque changement d'état
  - La création d'une transition s'effectue en définissant les quatre paramètres de base suivants :
    - l'identifiant de la transition
    - l'état de départ
    - l'état d'arrivée
    - l'effet de transition exécuté

Liste des effets de transition disponibles

Effet	Description
Move	Modifie la position d'un composant.
Rezise	Modifie la hauteur et la largeur d'un composant.
Rotate	Effectue la rotation d'un objet suivant un angle spécifique.
Animate	Permet d'animer certaines propriété de l'objet concerné (X, Y, alpha…).
AnimateColor	Réalise une animation basée sur le changement de couleur réalisé en parcourant le spectre entre deux couleurs définies.
AnimateFiltrer	Permet d'utiliser les filtres définis dans le package spark.filters sur le composant cible de l'animation.
AnimateTransitionShader	En spécifiant une image de début et une image de fin, il est alors possible d'utiliser cet effet afin de passer de l'une à l'autre.
CrossFade	Destiné aux images, il permet de passer d'une image à une autre de façon progressive.
Move 3D	Permet de faire bouger un objet sur l'ensemble des axes X, Y et Z.
Rotate 3D	Permet de réaliser des rotations d'objet sur les 3 axes de rotations (X, Y et Z).
Scale	L'effet Scale met à l'échelle un objet cible dans les directions X et Y autour du centre de transformation.
Scale3D	La classe Scale3D met à l'échelle un objet cible en 3 dimensions autour du centre de transformation.
WipeDirection	Effet de rideau en fonction de la direction donnée.

Exemple d'implantation d'effet

```
<s:transitions>
        <s:Transition id="transition" fromState="*" toState="*">
                <s:Wipe direction="down" duration="1000" target="{panel1}">
                </s:Wipe>
        </s:Transition>
</s:transitions>
```

- Le nom a été remplacé par un astérisque (*) signifiant que la transition concerne l'ensemble des changements d'états de l'application
- L'expression littérale de notre transition est la suivante : « Pour chaque changement d'état effectué dans l'application, appliquer l'effet **Wipe** ayant une durée de 1000 millisecondes à la cible : le composant **panel1**

Effet et cible (target)

- Chaque effet doit être appliqué sur une cible dont l'identifiant est passé en paramètre de la propriété target de l'effet à l'aide d'accolades ouvrante et fermante
- Pour appliquer l'effet à plusieurs cibles, il convient d'utiliser la propriété targets, à laquelle nous passons en paramètre la liste des identifiants des cibles concernées :

```
<s:Wipe direction="down" duration="1000"
   targets="{[panel1,numPatient]}">
</s:Wipe>
```

Dynamisme, Transitions, effets

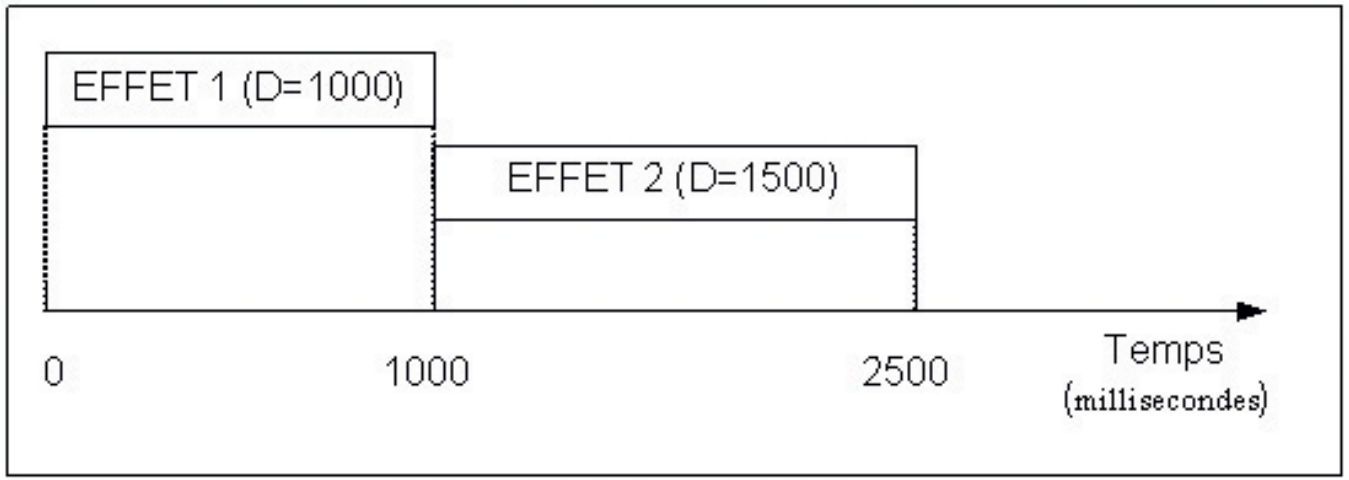
Les états ou view states

- Combiner des effets
  - Une transition peut comporter plusieurs effets ! Dans ce cas, comment ordonner leur exécution ?
  - On utilise les balises <s:Sequence> et <s:Parallel>

Dynamisme, Transitions, effets

Les états ou view states

- Exécution en séquence
  - La balise <s:sequence> exécute les effets de façon successive
  - Si la transition comporte deux effets, l'effet 1 sera exécuté, puis l'effet 2, à condition que l'exécution du premier effet soit terminée



Dynamisme, Transitions, effets

Les états ou view states

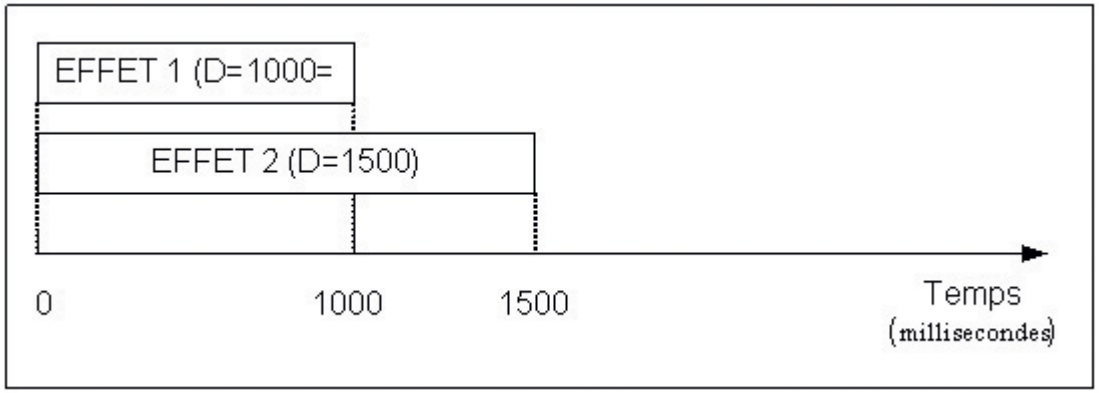
- Exécution en séquence : Exemple

```
<!-- TRANSITIONS -->
<s:transitions>
    <s:Transition id="transition" fromState="*" toState="*">
      <s:Sequence>
            <s:Wipe target="{panel1}" duration="1000"
            direction="down"/>
            <s:Resize target="{panel1}" duration="1000"
            widthFrom="0" widthTo="320"/>
      </s:Sequence>
    </s:Transition>
</s:transitions>
```

Dynamisme, Transitions, effets

Les états ou view states

- Exécution en parallèle
  - <s:Parallel> permet l'exécution d'effets de façon simultanée



Dynamisme, Transitions, effets

Les états ou view states

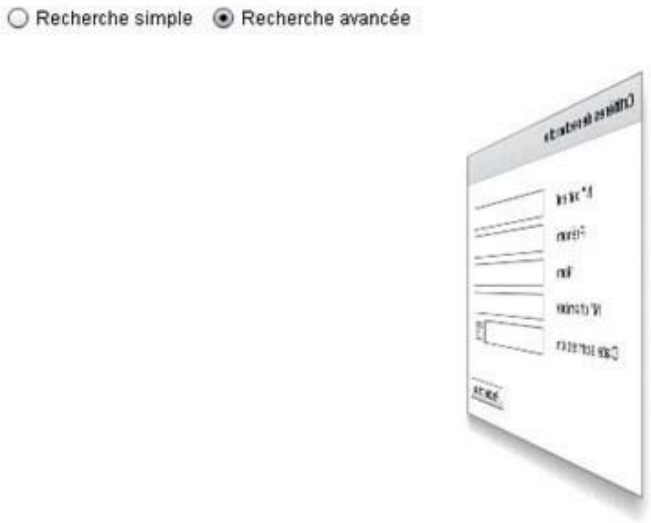
- Exécution en parallèle : transition1.mxml

```
<s:transitions>
    <s:Transition id=”transition” fromState=”*” toState=”*”>
      <s:Parallel>
            <s:Wipe target=”{panel1}” duration=”1000”
            direction=”down”/>
            <s:Resize target=”{panel1}” duration=”1500”
            widthFrom=”0” widthTo=”320”/>
      </s:Parallel>
    </s:Transition>
</s:transitions>
```

Dynamisme, Transitions, effets

Les états ou view states

- Autre exemple : Transtion2.mxml
  - Les effets choisis sont :
    - une rotation 3D complète du panneau sur l'axe X
    - un effet de déplacement du panneau vers la droite

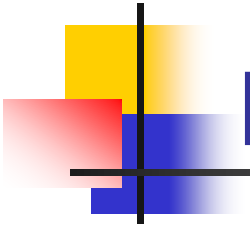


Dynamisme, Transitions, effets

Les états ou view states

- Exemple : Transtion2.mxml

```
<s:transitions>
    <s:Transition id=”transition” fromState=”*” toState=”*”>
      <s:Parallel>
            <s:Rotate3D target=”{panel1}”
            duration=”1000” angleYFrom=”0”
            angleYTo=”360”/>
            <s:Move target=”{panel1}” duration=”1000”
            xFrom=”0” xTo=”600”/>
      </s:Parallel>
    </s:Transition>
</s:transitions>
```



Dynamisme, Transitions, effets

- Les effets
  - Flex ajoute des effets lors des transitions
  - Exemple
    - Effet déplaçant l'objet vers la droite tout en le déplaçant vers le bas
    - repeatCount="4" : l'effet est réalisé 4 fois
    - repeatBehavior="reverse" : revient à l'état initial à la fin de chaque répétition (Reverse)
    - motionPaths : liste des propriétés à modifier (SimpleMotionPath)

Dynamisme, Transitions, effets

- L'effet Animate : Animate1.mxml
  - Permet d'animer les propriétés d'un objet cible, ici x et y

```
<fx:Declarations>
   <s:Animate id="Effet" target="{Cible}" repeatCount="4"
    repeatBehavior="reverse">
   <s:motionPaths>
         <s:SimpleMotionPath property="x" valueFrom="0"
         valueTo="100"/>
         <s:SimpleMotionPath property="y" valueFrom="0"
         valueTo="100"/>
   </s:motionPaths>
</s:Animate>
</fx:Declarations>
```

Dynamisme, Transitions, effets

Les états ou view states

```
<!-- Cible -->
   <s:Panel id="Cible" x="24" y="25"></s:Panel>
   <!-- Bouton d'exécution -->
         <s:Button x="374" y="25" label="Animer !"
         click="Effet.play()"/>
```

Dynamisme, Transitions, effets

Les états ou view states

- L'effet AnimateColor : AnimateColor.mxml
  - Permet de passer d'une couleur de base à une autre en parcourant le spectre qui les sépare
  - Dans l'exemple suivant, nous modifions la couleur de fond (backgroundColor) de l'objet cible en passant du blanc au noir

Dynamisme, Transitions, effets

Les états ou view states

- Exemple : AnimateColor.mxml

```
<fx:Declarations>
    <s:AnimateColor id="Effet"  target="{Cible}"
      colorPropertyName="backgroundColor" colorFrom="0xFFFFF"
      colorTo="0x00000" repeatCount="4" repeatDelay="1000"
      duration="2000">
    </s:AnimateColor>
</fx:Declarations>
<!-- Cible -->
<s:Panel id="Cible" x="24" y="25"></s:Panel>
<!-- Bouton d'exécution -->
<s:Button x="374" y="25" label="Animer !" click="Effet.play()"/>
```

Dynamisme, Transitions, effets

Les états ou view states

- L'effet AnimateFilter : AnimateFilter.mxml
  - Permet d'utiliser les filtres définis par l'ensemble des classes contenues dans le package spark.filters sur un objet de l'interface

Exemple : AnimateFilter.mxml

```
<fx:Declarations>
<!-- Effet permettant de réaliser un biseautage du panneau cible-->
<s:BevelFilter distance="5" angle="45" highlightColor="0xFFFF00"
    highlightAlpha=".8" shadowColor="0x0000FF" shadowAlpha=".8"
    blurX="3" blurY="3" strength="1"/>
<s:AnimateFilter id="Effet" target="{Cible}" duration="500"
    bitmapFilter="{new spark.filters.BevelFilter()}">
    <s:SimpleMotionPath property="distance" valueFrom="0"
    valueTo="10"/>
    <s:SimpleMotionPath property="angle" valueFrom="270"
    valueTo="360"/>
</s:AnimateFilter>
</fx:Declarations>
<!-- Cible -->
<s:Panel id="Cible" x="24" y="25"></s:Panel>
<!-- Bouton d'exécution -->
<s:Button x="374" y="25" label="Animer !" click="Effet.play()"/>
```

Les effets Move3D et Rotate3D : Move3D.mxml

- Permettent d'animer l'objet sur les trois axes de symétrie
- Dans cet exemple, nous exécutons de façon séquentielle les deux effets de rotation et de déplacement :

```
<fx:Declarations>
<s:Sequence id="Effet" target="{Cible}">
<s:Move3D id="move3D" duration="1000" xFrom="10"
  zTo="100" />
<s:Rotate3D id="rotate3D" duration="1000" angleYFrom="0"
  angleYTo="360" />
</s:Sequence>
</fx:Declarations>
```

- L’effet Scale3D : Scale3D.mxml
  - Permet de redimensionner les objets sur les 3 axes de symétrie :

```
<fx:Declarations>
<!-- Mise à l’échelle sur les 3 axes repeatBehavior="loop" : ne
   revient pas à l’état initial de la forme. L’objet sera
   redimensionné à chaque clique sur le bouton jusqu’à
   disparaître de l’écran
-->
<s:Scale3D id="Effet" target="{Cible}"
   scaleXBy="-.25"
   scaleYBy="-.25"
   scaleZBy="-.25"
   repeatBehavior="loop" />
</fx:Declarations>
```

- L’effet vitesse : Antoine-voitureRoulante.mxml, leroux.mxml

```
<fx:Declarations>
 <s:Animate id="GoAvant" target="{cible}" repeatCount="-1">
<s:motionPaths>
  <s:SimpleMotionPath property="x" valueFrom="300" valueTo="0"/>
</s:motionPaths>
</s:Animate>
<s:Animate id="GoArriere" target="{cible}" repeatCount="-1">
<s:motionPaths>
<s:SimpleMotionPath property="x" valueFrom="0" valueTo="300"/>
</s:motionPaths>
</s:Animate>
</fx:Declarations>

<s:Button x="20" y="70" label="Marche avant!" click="GoAvant.play()"/>
<s:Button x="150" y="70" label="Marche arrière!"
    click="GoArriere.play()"/>
```

Dynamisme, Transitions, effets

Les états ou view states

- Le package spark.effects.easing
  - contient les classes d'accélération pour les classes d'effets Spark

Classe	Description
Bounce	La classe Bounce implémente une fonctionnalité d'accélération simulant l'attraction gravitationnelle et le rebond de l'objet cible.
EaseInOutBase	La classe EaseInOutBase est la classe de base fournissant une fonction d'accélération.
EasingFraction	La classe EasingFraction définit des constantes pour la propriété easeInFraction de la classe EaseInOutBase.
Elastic	La classe Elastic implémente une fonctionnalité d'accélération dans laquelle le mouvement de l'objet cible est défini par une onde sinusoïdale qui décroît exponentiellement.
Linear	La classe Linear définit une accélération composée de trois phases : accélération, mouvement uniforme et ralentissement.
Power	La classe Power définit la fonction d'accélération à l'aide d'une expression polynomiale.
Sine	La classe Sine définit une fonction d'accélération à l'aide d'une fonction Sine.

- Test de la classe Elastic : voir d'abord Elastic.mxml
  - La classe Elastic implémente une fonctionnalité d'accélération dans laquelle le mouvement de l'objet cible est défini par une onde sinusoïdale qui décroît exponentiellement

```
<fx:Declarations>
   <s:Bounce id="bounceEasing"/>
   <s:Elastic id="elasticEasing"/>
   <s:Move id="moveRight" target="{myImage}" xBy="500"
           duration="2000" easer="{elasticEasing}"/>
   <s:Move id="moveLeft" target="{myImage}" xBy="-500"
   duration="2000" easer="{bounceEasing}"/>
</fx:Declarations>
```

- Suite1 : la cible : un logo dans un panel

```
<s:Panel id="examplePanel"
 title="Bounce and Elastic Effect Example"
 width="75%" height="75%">

<!-- Directions -->
<s:VGroup id="detailsBox" width="50%" top="5" left="5">
    <s:Label width="99%" color="blue"
    text="Click the buttons to watch the effect."/>
</s:VGroup>

<mx:Image id="myImage" top="20"
  source="@Embed(source='logoFlex.jpg')"/>
```

- Suite2 : boutons pour faire bouger la cible

```
<s:Button label="Move Right"
  bottom="10" left="5"
  click="moveRight.end();moveRight.play();"/>

<s:Button label="Move Left"
  bottom="10" left="100"
  click="moveLeft.end();moveLeft.play();"/>
</s:Panel>
```

Dynamisme, Transitions, effets

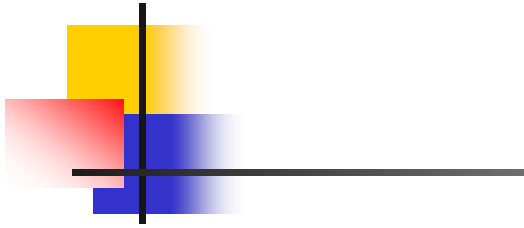
Les états ou view states

- Modifier le curseur de la souris
  - Il est parfois intéressant de pouvoir modifier le curseur de la souris afin de respecter la charte graphique préconisée par le client, et donner un côté un peu loufoque aux applications ou bien répondre à un besoin de sémantique fonctionnelle :
    - utilisation d'une loupe pour zoomer, d'une gomme pour effacer, etc.
  - Pour réaliser cette tâche
    - utiliser la méthode setCursor de la classe CursorManager en lui passant en paramètre une image qui aura préalablement été transformée en objet grâce à la classe Class :

- Modifier le curseur de la souris : ModifierCurseur.mxml

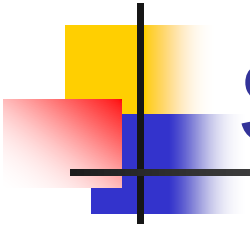
```
<fx:Script>
<![CDATA[
    public function changerCurseurSourisAvecImage():void{
    [Embed(source="LogoFlex.jpg")]
    var oLogoFlex:Class
    cursorManager.setCursor(oLogoFlex);
}
]]>
</fx:Script>
<s:Button x="35" y="24" label="Button"
    click="changerCurseurSourisAvecImage()"/>
</s:Application>
```

- Dans notre exemple,
  - Nous substituons le curseur de la souris par le logo de Flex lorsque nous cliquons sur le bouton



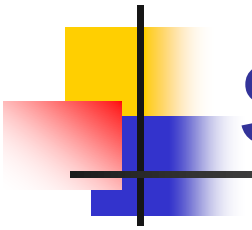
Flex 4

Notion de Skin (habillage)



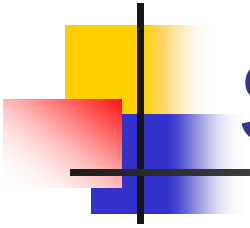
Skin

- Principe
  - Un des principaux changements de Flex 4 est la séparation de l'apparence (Skin) du comportement d'un composant
  - Avec Flex 3, la partie Skinning d'une application se résumait bien souvent à **modifier les styles de composants de base (styles CSS)**
  - Grâce à Flex 4, il est possible de faire des habillages personnalisés intéressants



Skin

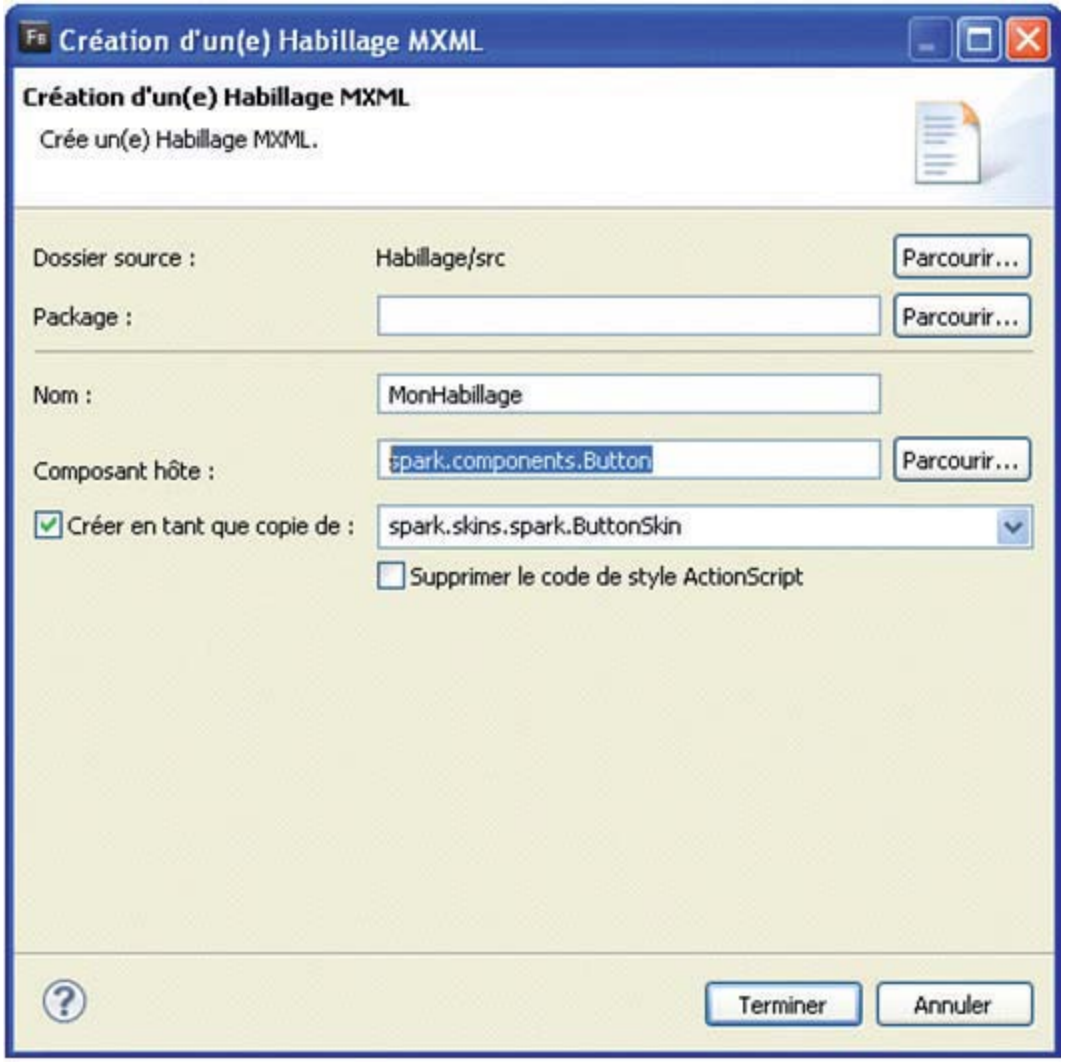
- Créer un habillage ou skin personnalisé
  - Définition
    - Un habillage (*skin) est un fichier MXML annexe à l'application principale, comportant l'ensemble des éléments définissant les propriétés graphiques d'un composant*
    - Il nous est alors possible de définir la couleur, la police, les contours et tout autre élément graphique supporté par le composant destinataire de l'habillage
    - Ainsi, on peut définir un comportement graphique sur un bouton lorsque celui-ci est affiché à l'écran puis un autre lorsque l'utilisateur clique sur ce dernier

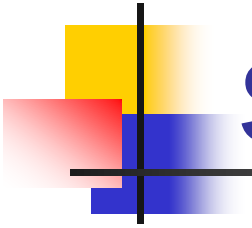


Skin

- Procédure
  - Pour ce faire, grâce à un simple clic droit sur le nom du projet, choisir l'option Nouveau puis Habillage MXML
  - Dans la fenêtre qui se présente à vous, saisir MonHabillage dans la zone Nom (correspondant au nom donné à notre habillage)
  - Comme nous souhaitons habiller les boutons à l'aide de notre production graphique, saisissons spark.components.Button dans la zone de texte composant hôte
  - Une fois ceci terminé, nous voyons dans l'arborescence de notre projet un nouveau fichier MonHabillage.mxml

Skin


Création d'un(e) Habillage MXML
Création d'un(e) Habillage MXML
Crée un(e) Habillage MXML.
Dossier source :
Habillage/src
Parcourir...
Package :
Parcourir...
Nom :
MonHabillage
Composant hôte :
spark.components.Button
Parcourir...
Créer en tant que copie de :
spark.skins.spark.ButtonSkin
Supprimer le code de style ActionScript
Terminer
Annuler



Skin

- Procédure
  - Ce fichier est découpé en parties fonctionnelles qui définissent l'habillage du composant bouton
  - Les différents états de notre composant sont les suivants
    - Up : état normal du bouton
    - Over : état du bouton lors du survol de la souris
    - Down : état du bouton lorsque l'utilisateur clique dessus
    - Disabled : état du bouton désactivé

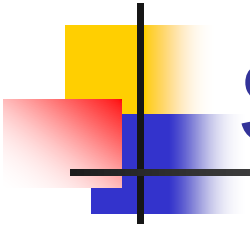
```
<?xml version="1.0" encoding="utf-8"?>
<s:Skin xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx" alpha.disabled=".5">

<s:states>
        <s:State name="up" />
        <s:State name="over" />
        <s:State name="down" />
        <s:State name="disabled" />
</s:states>

<s:Rect radiusX="4" radiusY="4" top="0" right="0" bottom="0"
        left="0" includeIn="down">
        <s:fill>
                <s:SolidColor color="0"/>
        </s:fill>
        <s:filters>
                <s:DropShadowFilter knockout="true" blurX="5"
                blurY="5" alpha="0.32" distance="2" />
        </s:filters>
</s:Rect>
```

```
<s:Rect id="rect" radiusX="4" radiusY="4" top="0"
        right="0" bottom="0" left="0">
        <s:fill>
                <s:SolidColor color="0x0099FF"
        color.over="0x0066FF" color.down="0x0000CC"/>
        </s:fill>
        <s:stroke>
                <s:SolidColorStroke color="0x222222" weight="2"/>
        </s:stroke>
</s:Rect>
<s:Rect radiusX="4" radiusY="4" top="2" right="2" left="2" height="50%">
        <s:fill>
                <s:LinearGradient rotation="90">
                        <s:GradientEntry color="0xFFFFFF"
                        alpha=".5"/>
                        <s:GradientEntry color="0xFFFFFF"
                        alpha=".1"/>
                </s:LinearGradient>
        </s:fill>
</s:Rect>
```

```
<s:Label text="Bouton" color="0x222222" textAlign="center"
         verticalAlign="middle" horizontalCenter="0"  verticalCenter="1"
         left="12" right="12" top="6" bottom="6"/>
</s:Skin>
```



Skin

- Exploiter l'habillage dans une application
  - Notre habillage étant terminé, voyons à présent comment l'utiliser dans une application Flex
  - Pour utiliser l'habillage que nous venons de créer sur un composant de type bouton, il suffit d'écrire cette ligne de code dans le fichier principal de l'application que vous avez créée en début de chapitre : ProjHabillage.mxml

```
<?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:Button horizontalCenter="0" verticalCenter="0"
   skinClass="{MonHabillage}"/>
</s:Application>
```