OpenLazslo

Yves Bekkers

Plateforme de développement pour des applications web

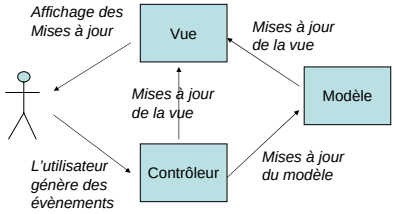
- RIA *Rich Internet Application*
  - Applications riches comprenant une interface riche en fonctionnalités (drag & drop, onglet, menu déroulant, animation etc.)
  - S'exécute sans aucune installation
  - S'exécute dans un bac à sable sûr, qui protège l'environnement local du client
  - L'information est stockée sur un serveur distant
  - L'application peut être lancée de partout
  - L'application est indépendante de la plateforme
  - Pour amélioré le confort et la disponibilité de l'UI, le client et le serveur échangent des informations en arrière plan

Jalons

- Laszlo Presentation Server (LPS)
  - 2000 premier prototype
  - 2001 début du développement
  - 2002 LPS pre-releases; premier déploiement Laszlo application (Behr)
  - 2003 LPS 1.0, 1.1 released; déploiement applications (Yahoo, Earthlink)
  - 2004 LPS 2.0, 2.1, 2.2 released; LPS devient un projet open source
- 2005 OpenLaszlo 3.0 released; changement de nom, Lazlo devient OpenLaszlo

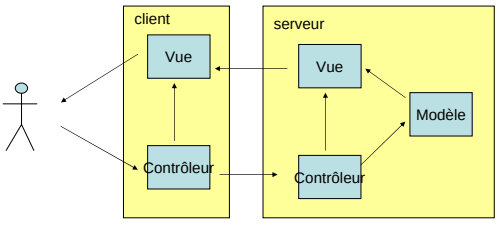
Architectures mvc

- Séparer les rôles dans un GUI



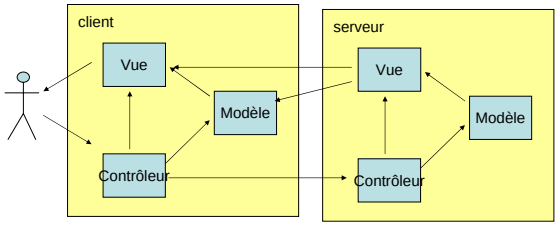
Comparaison des mvc web - 1

- Tous les calculs sont fait sur le serveur y compris la validation : JSF – Struts



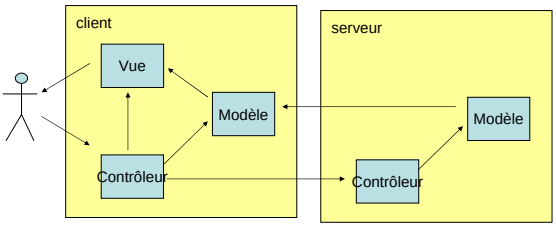
Comparaison des mvc web - 2

- Introduire un peu de logique chez le client,
- La validation en Javascript par exemple



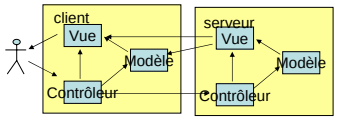
Comparaison des mvc web - 3

- Déporter toutes les fonctionnalités de la vue chez le client : c'est le principe des RIA

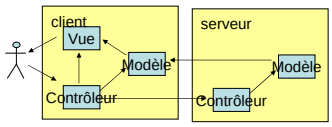


Comparaison des deux modèles

- Double vue
- Cycle request/réponse complêt
- Lourde charge pour le serveur
- Lourde charge pour le réseau



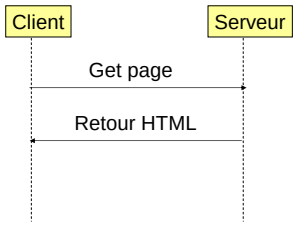
- Une seule vue chez le client
- request/réponse si nécessaire
- Interface client riche
- Charge chez le client
- Allège la charge du serveur
- et celle du réseau



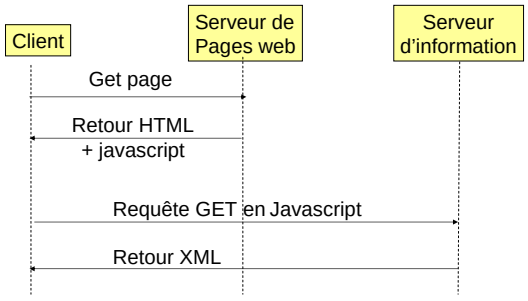
Fonctionnement asynchrone

- Comme les architectures AJAX
  - Aspect « *Asynchronous JavaScript* »
- Un unique chargement de page
  - Envoie de requêtes au serveur sans rechargement de page
  - Mise à jour de portions de page sur réception d'évènements utilisateur
  - Limites les transferts de communications entre serveur et client

Client serveur classique



Client serveur asynchrone



OpenLaszlo

- Programmation objet +programmation déclarative
  - Liaison déclarative des données XML aux composants de l'interface
  - Un langage déclaratif de description d'interfaces
- D'énormes capacités graphique : Flash
  - Graphique vectoriel
  - Données multimédia : images, son, vidéo
  - Streaming
- Un outil de conception d'application « wysiwyg » « open source » Eclipse

Aspect graphique, ressources multimédia, …

- Tous les avantages de flash pour
  - Le graphique vectoriel
  - La gestion des images
  - La gestion du son
  - La gestion des vidéos
  - Les animations
- L'inconvénient de flash (pour le client)
  - Il faut un plugin flash
- Mais avantage sur d'autres technologies flash
  - On fait du flash sans rien connaitre à flash

Caractéristiques

- OpenLaszlo utilise
  - **Lzx** un langage de composants d'interface
    - L'élément racine est `<canvas>` que l'on peut rapproché de l'élément `<html>` pour html
  - **JavaScript** pour décrire les interactions entre composants et entre le clients et le serveur
  - **XML** pour conserver/échanger les données
  - **Flash** (bientôt DHTML) pour le graphisme, le dynamisme et le multimédia

Spécification d'interfaces

- *Langage de spécification d'interfaces : lzx*
  - Une interface est décrite par un document XML contenant du code Javascript.
- *Un compilateur d'interface*
  - traduit les documents LZX en un programme flash Macromédia.
- Un serveur J2EE (conteneur de servlet)

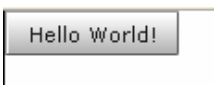
Quelques exemples introductifs

L'application « hello word »

```
<canvas width="200" >
  <text>Hello World!</text>
</canvas>
```


Un bouton

```
<canvas width="200">
  <button>Hello World!</button>
</canvas>
```



Une fenêtre avec un bouton

```
<canvas width="200">
  <window x="10" y="10" width="150"
        height="150" >
    <button>Hello World!</button>
  </window>
</canvas>
```

Un formulaire

```
<canvas  width="240">
  <dataset name="echoer" src="http:echo.jsp"/>

  <window x="10" y="10" >
   <form id="ex1" bgcolor="0xAAB5C8" >
     <submit name="submitter«
        data="${echoer}"/>

      <statictext fontstyle="bold" >
        What's your current name?
      </statictext>
      <edittext width="175" />

      <statictext fontstyle="bold" >
        What's in a name?
      </statictext>
      <radiogroup x="25" name="definition">
        ...
```

Les composants de base

Les éléments <cavas> <view>

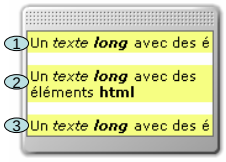
```
<canvas width="200">
  <view width="100" height="100"
        bgcolor="red"/>
</canvas>
```

Insersion d'une image dans une vue <view>

```
<canvas width="200">
  <view resource="cd_cover.jpg"/>
</canvas>
```

Labels : textes non éditables

```
① <text width="150">
     Un <i>texte <b>long</b></i> avec des
     éléments <b>html</b>
  </text>
② <text width="150" multiline="true"”> ...
③ <statictext width="150"> ...
```



Éléments HTML autorisés

<a>Lien hypertexte

Retour chariot
<b>Texte en gras
<font>Définir et utiliser des fonts
<i>Italique
<p>Paragraphe
<u>Souligné
<img>image (Flash Player 7 seulement)

Changement de police de caractères

```
<font name="helmet" src="helmetb.ttf"/>

<text font="helmet" fontsize="20"
     fgcolor="#175DDE">
     Where to Begin
</text>
```

Where to Begin

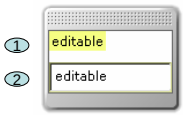
Textes éditables

- Look & feel redéfinissable

① `<inputtext>editable</inputtext>`

- Look & feel fixé (une boite de saisie)

② `<edittext>editable</edittext>`



<inputtext>

```
<inputtext width="150">
     Bonjour !
     et patatis et patata, ceci
     pour vous montrer un texte
     d'un certaine longueur.
</inputtext>
```

Sans `multiline="true"`

Bonjour ! et patatis et pat

Avec `multiline="true"`

Bonjour ! et patatis et patata, ceci pour vous montrer un texte d'une certaine longueur.

<combobox>

```
<combobox id="cbox" x="5" y="5"
       width="130" editable="false">
  <textlistitem text="Green" value="green"/>
  <textlistitem text="red" value="red"/>
  <textlistitem text="Light Blue"
                value="0xaaddff"/>
</combobox>
<view x="150" y="5" width="20" height="20"
          bgcolor="${cbox.value}"/>
```

Implantation des composants

```
<canvas width="200">
  <simplelayout axis="y" spacing="4"/>
  <text>Where to Begin</text>
  <view resource="cd_cover.jpg"/>
</canvas>
```

<vbox> <hbox>

```
<view>
   <simplelayout/>
</view>
```

```
<vbox>
   ...
</vbox>
```

```
<view>
   <simplelayout axis="x"/>
</view>
```

```
<hbox>
   ...
</hbox>
```


Barres de défilement

```
<canvas width="200">
  <view clip="true" width="200" height="200">
    <view resource="culebra-trip.jpg"/>
    <scrollbar axis="x"/>
    <scrollbar axis="y"/>
  </view>
</canvas>
```

Ressources audio et vidéo

```
<canvas width="500" height="800">
  <simplelayout/>
  <view resource="luckyluke_anime.gif"/>
  <resource name="univRennes1“
     src="logo-UNIV-Rennes-1.gif"/>
  <view resource="univRennes1"/>
</canvas>
```

Modularité

```
<canvas width="100%" height="100%">
   <include href="helloworld.lzx"/>
</canvas>
```

```
<library>
  <window>
    <text>Hello World.</text>
  </window>
</library>
```


Les documents <library>

- Ne sont inclus qu'une seule et unique fois

```
<canvas width="100%" height="100%">
   <include href="helloworld.lzx"/>
   <include href="helloworld.lzx"/>
</canvas>
```

Le second est sans effet !

Inclure des documents xml

```
<button  text="New"/>
```

bouton.xml

- Tous les éléments <include> donnent lieu à inclusion

```
<canvas width="100%" height="100%">
   <include href="bouton.xml"/>
   <include href="bouton.xml"/>
</canvas>
```

Graphique

`<drawview>`

Tracer des rectangles

- rec(x,y,width,height[,radius])

```
<canvas debug="true">
   <drawview width="400" height="400">
      <handler name="oninit">
            this.rect(10, 10, 100, 30);
            this.rect(10, 50, 100, 30, 5);
            this.rect(10, 90, 100, 30, 15);
            this.stroke();
       </handler>
   </drawview>
</canvas>
```

Tracer un triangle

```
<canvas debug="true">
   <drawview width="400" height="400">
      <handler name="oninit">
            this.beginPath();
            this.moveTo(20,20);
            this.lineTo(75, 20);
            this.closePath();
            this.stroke();
       </handler>
   </drawview>
</canvas>
```

Tracer un cercle

- arc(x, y, radius, startAngle, endAngle, clockwise)

```
<canvas debug="true">
   <drawview width="400" height="400">
      <handler name="oninit">
         this.arc(150, 150, 100, 0, 180, false);
         this.strokeStyle = 0x0000ff;
         this.lineWidth = 1;
         this.stroke();
      </handler>
   </drawview>
</canvas>
```

Courbes de bézier

```
this.beginPath();
this.moveTo(75,40);
this.bezierCurveTo(75,37,70,25,50,25);
this.bezierCurveTo(20,25,20,62.5,20,62.5);
this.bezierCurveTo(20,80,40,102,75,120);
this.bezierCurveTo(110,102,130,80,130,62.5);
this.bezierCurveTo(130,62.5,130,25,100,25);
this.bezierCurveTo(85,25,75,37,75,40);
this.fill();
```

Programmation objet

Programmation objet – le problème

```
<canvas width="200">
  <simplelayout spacing="5"/>

  <view height="100" width="100"
 bgcolor="red"/>
  <view height="100" width="100"
 bgcolor="red"/>
  <view height="100" width="100"
 bgcolor="red"/>
</canvas>
```

Programmation objet – la solution

```
<canvas width="200">
  <simplelayout spacing="5"/>

  <class name="box" bgcolor="red"
    height="100" width="100"/>

  <box/>
  <box/>
  <box/>
</canvas>
```

Déclaration

Constructeurs

Programmation objet – Héritage

```
<canvas width="200">
  <simplelayout spacing="5"/>

  <class name="box" bgcolor="red"
    extends="view"
    height="100" width="100"/>

  <box/>
  <box/>
  <box/>
</canvas>
```

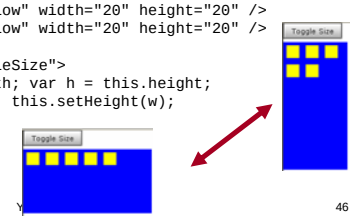
Héritage implicite

Définition de méthodes

```
<canvas width="200">
  <button onclick="vw.toggleSize()" text="Toggle Size"/>

  <view id="vw" y="30" width="100" height="200" bgcolor="blue">
    <wrappinglayout spacing="10" xinset="5" yinset="5" />
    <view bgcolor="yellow" width="20" height="20" />
    <view bgcolor="yellow" width="20" height="20" />
    <view bgcolor="yellow" width="20" height="20" />
    <view bgcolor="yellow" width="20" height="20" />
    <view bgcolor="yellow" width="20" height="20" />

    <method name="toggleSize">
      var w = this.width; var h = this.height;
      this.setWidth(h); this.setHeight(w);
    </method>
  </view>
</canvas>
```



Élément <attribute>

- Déclaration d'attributs d'instance

```
<canvas>
  <class name="val" extends="window"
    width="50" height="60">
    <attribute name="valeur" value="25"/>
    <text text="${parent.valeur}"/>
  </class>
  <val/>
</canvas>
```

Valeur contrainte

Valeur d'attribut indéfinie

- Déclaration d'un attribut

```
<attribute name="val"/>
```

- Test valeur indéfinie

```
if (typeof(val) == 'undefined') {
    ...
}
```

Construction d'instances

- Déclaration

```
<view id="monParent">
  <class name="maclass">
    <attribut name="val"/>
  </class>
<view>
```

- Création d'instance en LZX

```
<maclass val="15"/>
```

- Création d'instance en Javascript

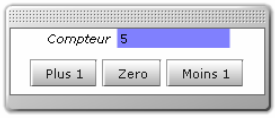
```
new maclass(monParent,{'val':15});
```


Un exemple d'interface contrainte

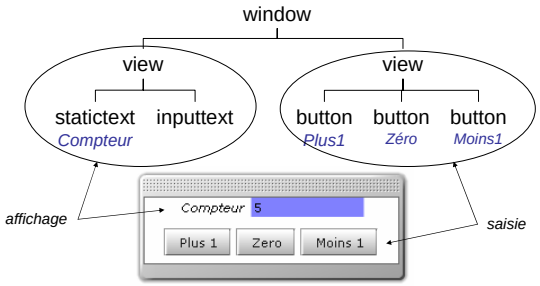
Utilisation des classes

Le problème du compteur

- Implémenter un compteur avec trois boutons,
  - `Plus1, Zéro, Moins1`



Structure de l'interface



La classe `compteur`

```
<class name="compteur"
  extends="window" width="244"
  height="102">
  <attribute name="val" value="0"/>
  <simplelayout axis="y" spacing="10"/>
  // affichage
  // saisie
</class>
```


Affichage

```
<view align="center">
  <simplelayout axis="x" spacing="4" />
  <statictext><i>Compteur</i></statictext>
  <inputtext maxlength="10"
    bgcolor="0x8080ff"
    text="${parent.parent.val}" />
</view>
```

Saisie

```
<view align="center">
  <simplelayout axis="x" spacing="4" />
  <button text="Plus 1">
    <method event="onclick">
    parent.parent.setAttribute('val',parent.parent.val+1);
    </method>
  </button>
  <button text="Zero">
    <method event="onclick">
    parent.parent.setAttribute('val',0);
    </method>
  </button>
  <button text="Moins 1">
    <method event="onclick">
    parent.parent.setAttribute('val',parent.parent.val-1);
    </method>
  </button>
</view>
```


Création dynamique d'un composant

```
<button id="myButton"
     text="Créer un compteur !"
     onclick="new compteur(canvas,{})"
/>
```

- Forme générale la plus répandue

`new nom(parent, params)`

  - `parent` nom du composant graphique parent ou null
  - `params` est une table de hash ou null
    `{'para1': val1, 'para2': val2, …}`
- Le nom d'un composant c'est le nom de sa classe sauf pour certains composants du langage lzx
  - Ainsi le nom de `<text>` est `LzText`

Programmation par évènements

Gestion d'évènements - exemple

```
<canvas width="200">
  <button onclick="vw.setWidth(200)"
          text="Expand Width" />
```

Nommage d'une vue

```
  <view id="vw" bgcolor="blue"
        y="30" width="100" height="100" >
    <method event="onwidth" >
      this.setBGColor(0xFF0000)
    </method>
   </view>
</canvas>
```

Programmation par évènements

- Trois sortes d'objets
  - Évènements
    - `<event> : LzEvent`
  - Délégués (écouteurs)
    - `<delegate> : LzDelegate`
    - Fonctions à enregistrer auprès des évènements
    - Des délégués pour les évènements connus statiquement
      - `<handler>, <method event=" … "/>`
  - Timer `LzTimer`
    - Pour exécuter du code après un certain délai

Les évènements

- Trois sortes d'évènements
  - Évènements liés à la souris
    - `onclick, ondblclick, onmouseover, onmouseout, …`
  - Évènements liés à l'instanciation des attributs
    - Pour tout attribut de nom `xxx` un évènement `onxxx` est généré pour tout changement de valeur de l'attribut fait par l'intermédiaire de la fonction `setAttribute('xxx',val)`
  - Évènements créés par l'utilisateur

Les évènements connus statiquement sur une <view>

onaddsubresource
onaddsubview
onblur
onclick
ondblclick
onerror
onfocus
onframe
onheight
onlastframe
onload
onmousedown
onmousedragin
onmousedragout
onmouseout
onmouseover
onmouseup
onmouseupoutside
onopacity
onplay
onremovesubview
onstop
ontimeout
onwidth
onx
ony

Évènements créés par l'utilisateur

- Déclaration
  ```
  <event name="myevent"/>
  ```
- Envoi d'un évènement
  ```
  this.myevent.sendEvent();
  ```


Quatre méthodes d'écoute

1. Par un attribut onXXX de composant
2. Par un élément <handler name="…" >
3. Par un élément <method event="…">
4. Par une contrainte ${…} dans un d'attribut

Écouter des évènements - 1

- Par un attribut onXXX de composant

```
<view onmouseover="doSomething()">

  <method name="doSomething"
    // code à exécuter lorsque la souris
    // est au dessus du composant
  </method>
</view>
```


Écouter des évènements - 2

- Par un élément <handler name="…" >

```
<view>
 <handler name="onmouseover">
    // code à exécuter lorsque la souris
    // est au dessus du composant
 </handler>
</view>
```

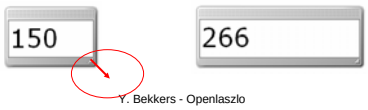

Écouter des évènements - 3

- Par un élément <method event="…">
  ```
  <view>
   <method event="onmouseover">
      // code à exécuter lorsque la souris est
      // au dessus du composant
   </method>
  </view>
  ```
- *Attention* ! ne pas confondre
  - Définition d'une méthode <method **name**="…">
  - Définition d'un handler <method **event**="…">
  - Les attributs **@event** et **@name** sont exclusifs l'un de l'autre

Écouter des évènements - 4

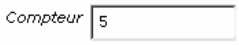
- Par une contrainte ${…} dans un d'attribut

```
<window width="50" height="60"
                        resizable="true">
   <attribute name="val" value="${width}"/>
   <text text="${parent.val}"/>
</window>
```



<edittext> évènement onkeydown

- Le composant <edittext> permet de saisir des textes



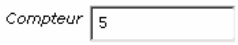
- On se met à l'écoute de l'évènement onkeydown

```
<edittext text="${'Inc '+inc}">
   <handler name="onkeydown" args="k">
      if (k == 13) { // RC = 13
         … sauver la valeur …
      }
   </handler>
</edittext>
```

Le code de la clé est passé en paramètre par le système

<edittext> évènement onkeydown (bis)

- On récupère la valeur par la méthode
  - getValue()
- Il s'agit d'un texte, si nécessaire on peut le convertir en un entier par l'expression
  - 1*getValue();



Composant générique <edittext>

- Récupération du texte sur « retour chariot »

```
<class name="edittextRC" extends="edittext">
   <handler name="onkeydown" args="k">
      if (k == 13) { // RC = 13
         saveValue();
      }
   </handler>
</class>
```


Composant générique - utilisation

- Déclaration

```
<class name="edittextRC" extends="edittext">
   <handler name="onkeydown" args="k">
      if (k == 13) {saveValue();}
   </handler>
</class>
```

Redéfinition de la méthode saveValue()

- Utilisation

```
<edittextRC text="${parent.currentInc}">
   <method name="saveValue">
      parent.setAttribute('currentInc',
                          1*getValue());
   </method>
</edittextRC>
```


Objet LzDelegate

- L'objet LzDelegate est un pointeur de fonction qui donne accès à une fonction dans un contexte (i.e. une instance de classe).
- Lorsque sa méthode execute() est appelé c'est la fonction qu'il repère qui est exécutée
- Un délégué s'enregiste en général au prés d'un event mais il peut aussi s'enregistrer au prés d'un timer
- Constructeur

```
new LzDelegate(this, "methodName",// objet-method
       eventSender, "eventName")  // émetteur-évènement
```

Deux méthodes de création

- Construction « tout en un »

```
new LzDelegate(this, "methodName", // objet-method
       eventSender, "eventName") // émetteur-évènement
```

- Construction en deux temps

```
var del = new LzDelegate(this, "methodName ");
          // objet-method
del.register(eventSender, "eventName")
          // émetteur-évènement
```


Relations entre LzDelegate et Event

- Emetteur et évènement connu statiquement

```
<handler name="eventName" reference="eventSender">
 ... faire quelque chose ...
</handler>
```

- Emetteur et évènement inconnu statiquement

```
<method name="methodName">
 ... faire quelque chose ...
</method>
<method name="createDelegate>
  this.myDel = new LzDelegate(this, "methodName“);
  this.myDel.register(eventSender, "eventName");
</method>
```


Les attributs handlers d'évènement

- Les attributs onXXX (handlers implicites)

```
onclick="do something or other";
```

- S'étendent en

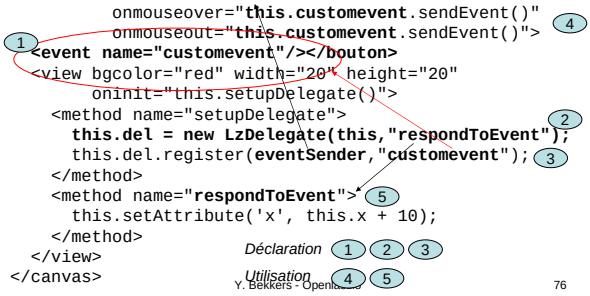
```
<handler name="onclick" >
  … do something or other;
</handler>
```

- Qui s'étendent à nouveau en

```
// créer un nom de méthode unique
var uid = $m + <unique id>
// définir une méthode
this.$m = function () { ; };
// créer un délégué
var del = new LzDelegate(this, $m);
// enregistrer le délégué
del.register(objet, 'onclick');
```


Evènements de l'utilisateur

```
<canvas height="40">
  <simplelayout/>
  <button name="eventSender"
          onmouseover="this.customevent.sendEvent()"
          onmouseout="this.customevent.sendEvent()">
    <event name="customevent"/></bouton>
    <view bgcolor="red" width="20" height="20"
          oninit="this.setupDelegate()">
      <method name="setupDelegate">
        this.del = new LzDelegate(this,"respondToEvent");
        this.del.register(eventSender,"customevent");
      </method>
      <method name="respondToEvent">
        this.setAttribute('x', this.x + 10);
      </method>
  </view>
</canvas>
```



LzTimer

- Exécuter du code après un certain délai

```
this.del = new LzDelegate(this,"do");
LzTimer.addTimer(this.del, 1000);
```

- La méthode `do()` sera exécuter après 1s de délai

Data, XML, XPath

Manipulation de données XML

- LZX facilite l'accès aux données structurées via XML
- LZX facilite la réalisation des interfaces en autorisant la mise en relation directe des données à l'interface utilisateur
- Les outils proposés sont :
  - Embarquer des données structurées dans l'application
  - Créer et manipuler des données à l'exécution
  - Importer ou exporter de manière asynchrone, à l'exécution, des données à un serveur distant ou à un web service
  - Lier déclarativement aussi bien que dynamiquement les données à l'interface

Élément `<XMLHTTPRequest>`

- Implémente la requète `XMLHTTPRequest`, le « *pilier* » des applications Ajax
- Comforme à la spécification du consorsium Whatwg
  - « *Web Hypertext Application Technology Working Group* »
  - `http://www.whatwg.org/`
- Usage simplifié dans Openlazslo par l'emploi de l'élément `<dataset>`

Élément `<dataset>`

- Embarquer un document XML au chargement de l'application

```
<dataset name="contact">
   <contact>
      <name>John Doe</name>
      <phone>555-2000</phone>
   </contact>
</dataset>
```

- L'élément `dataset` associe un nom à un document XML

Chargement d'un `<dataset>`

- Deux modes de chargement
  - Statique : au chargement de l'application
    - Cas où il n'y a pas d'attribut `@src`
    - Cas où l'attribut `@src` contient un chemin local à l'application
  - Dynamique : par lancement de requêtes HTTP asynchrones
    - Cas où l'attribut `@src` contient une URL
    - `<dataset name="mondata" src="dd.xml" request="true"/>`

Problème du chargement statique

- Les `dataset` statiques sont compilés dans l'application.
  - Les évènements `ondata` de tout pointeur `LzPointer` référençant un `dataset` statique sont générés avant l'initialisation des composants graphiques
  - Leur code `ondata` ne doit pas référencer de composant graphique.
- Solution
  - Mettre le code d'initialisation sur l'évènement on init du composant à initialiser.

Liaison des composants de l'interface aux données d'un `<dataset>`

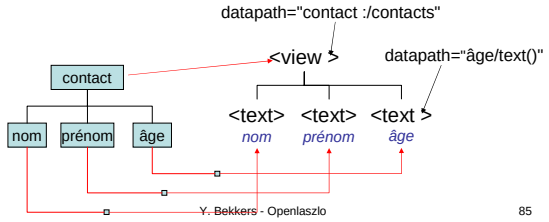
```
<dataset name="contact">
   <contact>
      <name>John Doe</name>
      <phone>555-2000</phone>
   </contact>
</dataset>
<view datapath="contact:/">
   <simplelayout axis="y" />
   <view datapath="contact/">
     <simplelayout axis="x" />
     <text datapath="name/text()" />
     <text datapath="phone/text()" />
   </view>
</view>
```

Expressions XPath

John Doe 555-2000

L'attribut @datapath

- Les attributs @datapath sont utilisés pour lier les arborescences de données XML aux arborescences de composants de l'interface par des chemins XPath



Données multiples

```
<canvas>
   <dataset name="contactInfo">
      <contacts>
         <contact>
            <name>John Doe</name>
            <phone>555-2000</phone>
            <DOB>16-01-1973</DOB>
            <role>Developer</role>
         </contact>
         <contact>
            <name>Jane Doe</name>
            <phone>555-3000</phone>
            <DOB>27-02-1974</DOB>
            <role>Manager</role>
         </contact>
      </contacts>
   </dataset>
...
```


Éléments répétés

```
<view datapath="contactInfo:/contacts/">
      <simplelayout axis="y" />
      <view datapath="contact/">
         <simplelayout axis="x" />
         <text datapath="name/text()" />
         <text datapath="phone/text()" />
         <text datapath="DOB/text()" />
         <text datapath="role/text()" />
      </view>
   </view>
</canvas>
```

- *élément répété autant que nécessaire*

John Doe	555-2000	16-01-1973	Developer
Jane Doe	555-3000	27-02-1974	Manager

Combobox liées à du XML - 1

```
<dataset name="items">
  <item value="item1">item one</item>
  <item value="item2">item two</item>
  <item value="item3">item three</item>
  <item value="item4">item four</item>
 </dataset>
```


Combobox liées à du XML - 2

```
<combobox x="5" y="5" width="130"
   shownitems="3" defaulttext="choose ...">
   <textlistitem datapath="items:/item"
      text="$path{'text()'}"
      value="$path{'@value'}"/>
</combobox>
```


Requètes Xpath dans le programme

```
<MyClass datapath="."
  ondata="setAttribute('MyAttr',
    datapath.xpathQuery('@boundary'))"
/>
```

- Requète relative à la position de l'attribut @datapath

Élément `datapointer`

- Déclaration du pointeur

```
<datapointer id="theContent"
  xpath="content:/content"
  ondata="initFeatures()">

  <method name="initFeatures">
    setAttribute('defaultScale',
    theContent.xpathQuery('content/@attr'));
  </method>
</datapointer>
```

Requête relative au pointeur sans déplacement du pointeur

Mouvements d'un pointeur

- L'élément `<dataPointer>` permet de parcourir le document XML
  - `selectChild()`
  - `selectChild(n)`
    - Descendre dans la hiérarchie de 1 ou n niveaux, donner le premier fils du niveau choisi
    - Rends vrai si le déplacement a été possible
  - `selectNext()`
  - `selectNext(n)`
    - Aller au 1ier ou nième frère qui suit
    - Rends vrai si le déplacement a été possible

Mouvements d'un pointeur (bis)

- Autres fonction de parcours
  - `selectParent()`
  - `selectParent(n)`
    - Monter dans la hiérarchie de 1 ou n niveaux
    - Rends vrai si le déplacement a été possible
  - `selectPrev()`
  - `selectPrev(n)`
    - Aller au 1ier ou nième frère qui précède.
    - Rends vrai si le déplacement a été possible

Parcours itératif des fils d'un élément

- Déclaration du pointeur

```
<datapointer id="pt"
    xpath="content:/content"/>
```

- Parcourir les fils

```
pt.selectChild();
do {
  var min = pt.getNodeAttribute('min');
  var max = pt.getNodeAttribute('max');
  Debug.write(min+" "+max);
} while (pt.selectNext());
```


Parcours récursif des fils d'un nœud

```
<method name="parcoursChild" args="pt">
  var hasChild = pt.selectChild();
  if (hasChild)
    do {
      Debug.write(pt.getNodeName());
      var features =
          parcoursChild(pt.dupePointer());
    } while (pt.selectNext());
</method>
```

- Appel (*copie du pointeur pour ne pas le détruire*)

```
parcoursChild(theContent.dupePointer());
```


Ajouter un nœud à un dataset

- Initialiser un pointeur

```
var dp = new LzDatapointer();
dp.setXPath('myData:/');
```

- Création directe d'un nœud avec contenu textuel

```
dp.addNode('incr',15,{});
```

- Création d'un nœud quelconque

```
var textNode = new LzDataText('16');
var lesFils = new Array(textNode);
var elem = new LzDataElement('incr',
  {},lesFils);
dp.p.appendChild(elem);
```

Réinsertion de <dataset>

- Un dataset peut être repeuplé à l'exécution en appelant sa méthode dorequest()
- Le résultat de la requête est retourné à l'application et l'évènement ondata du dataset est généré
- Les évènements ondata des datapointer liés au dataset seront eux aussi générés

Divers

- Les tableaux, les arbres
- Openlazslo et la mise au point
- Un problème spécifique de développement des RIA
- Mise en place d'un service d'impression pour l'application

Les tableaux, les arbres

Éléments <grid>

```
<canvas debug="true">
  <dataset name="users">
    <users>
      <personne>
        <nom>Durant</nom>
        <prenom>Paul</prenom>
        <age>35</age>
      </personne>
      <personne>
        <nom>Dupont</nom>
        <prenom>Jean</prenom>
        <age>50</age>
      </personne>
    </users>
  </dataset>
```

Deux personnes

Éléments <grid> (suite)

```
<view x="20" y="20">
  <grid name="grid_users" datapath="users:/users">
    <gridtext datapath="nom/text()">Nom</gridtext>
    <gridtext datapath="prenom/text()">Prénom</gridtext>
    <gridtext datapath="age/text()">Age</gridtext>
  </grid>
 </view>
</canvas>
```

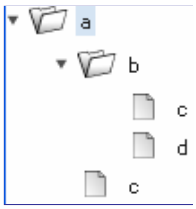
Nom	Prénom	Age
Durant	Paul	35
Dupont	Jean	50

Élément <tree> statique

```
<view width="200" height="200">
   <tree open="true" text="menu">
      <tree text="frites" isleaf="true"/>
      <tree open="true" text="salade">
         <tree text="carottes" isleaf="true"/>
         <tree text="laitue" isleaf="true"/>
         <tree text="oignons" isleaf="true"/>
         <tree text="pommes de terre" isleaf="true"/>
      </tree>
      <tree open="true" text="boisson">
         <tree text="vin" isleaf="true"/>
      </tree>
   </tree>
</view>
```

Arbre construit sur des données xml - 1 -

```
<dataset name="data">
   <a>
      <b>
         <c/>
         <d/>
      </b>
      <c/>
   </a>
</dataset>
```



Arbre construit sur des données xml - 2 -

```
<view width="200" height="200"
    datapath="data:/" >
  <simplelayout/>
  <tree datapath="*"
    text="$path{'name()'}"
    isleaf=
       "${!datapath.xpathQuery('*')}"
  />
</view>
```


Mise au point

Option debug

```
<canvas debug="true" width="200">

  <button id="myButton" text="Click Me!"
 onclick="Debug.write('Button Clicked',this)"/>

  <view id="logo" x="10" y="20"
  resource="logo.swf" visible="false"/>
</canvas>
```

Visualisation des traces

Évaluation d'expressions

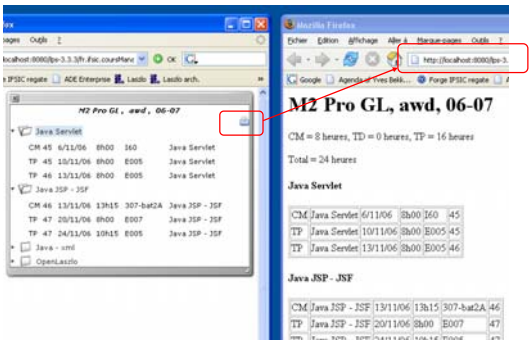
Expliciter la fenêtre de debug

```
<canvas debug="true" width="200">
  <debug width="300" height="60"/>
  <button id="myButton" text="Click Me!"
 onclick="Debug.write('Button Clicked',this)"/>

  <view id="logo" x="10" y="20"
  resource="logo.swf" visible="false"/>
</canvas>
```

Le problème du service d'impression

Exemple de service d'impression



`LzBrowser.loadURL()`

- Transformer les documents XML en HTML par un feuille de style XSLT
- Lancer une nouvelle fenêtre de serveur sur le résultat

```
<view datapath="." resource="../images/print_edit.gif"
   align="right"
   ondblclick="LzBrowser.loadURL('../xslt/xslt.jsp?'+
   'source='+parent.parent.appId+'/data/'+canvas.currentFile+
   '&amp;style='+parent.parent.appId+'/xslt/mesCours.xslt'+
   '&amp;FORMATION='+datapath.xpathQuery('@formation')+
   '&amp;TITRE='+datapath.xpathQuery('@titre')+
   '&amp;ANNEE='+datapath.xpathQuery('@année'),'_blank');"
/>
```


Le problème des tests

Déplacement dans l'interface
En phase de mise au point

Méthodologie de développement

- Applications WEB traditionnelle plusieurs points d'accès
- RIAs un seul point d'accès
  - Alors que l'application grossit, de plus en plus d'actions sont nécessaires pour atteindre la partie en cours de développement après une recompilation

Utiliser la modularité

- Mettre chaque composant complexe dans une librairie

```
<library>
  <window>
    <text>Hello World.</text>
  </window>
</library>
```

- Créer un emballage <canvas> pour chaque librairie

```
<canvas>
  <include href="helloworld.lzx"/>
</canvas>
```


Les avantages de la méthode

- Chaque emballeur peut être accédé individuellement pour se concentrer sur le développement de la partie emballée
- Des initialisations peuvent être faites dans l'emballeur pour tester différentes situations
- Les librairies ainsi crées sont rassemblées dans un unique <canvas> pour créer l'application complète
- On est ainsi prêt pour l'arrivé des librairies dynamiques …

Conclusions

Conclusions

- Avantage d'Openlaszlo
  - Open-source, stable, bien documenté
  - Indépendant de la plate forme de développement
  - Un environnement de développement extrêmement confortable
  - Langage orienté objet à base de composants d'IU
  - Étends les principes des RIA au multimédia
  - Utilise Flash
  - Pas de limite due au navigateur
- Inconvénients
  - Utilise Flash …
  - Indexation par les sites de référencement impossible
  - Les boutons de navigation du navigateur sont inutilisables
  - Pas de « printeur » intégré

Exemples de site

- **Amazon Store**
  - **Une boutique en ligne (cd musicaux) qui présente dans une même fenêtre les nombreuses fonctionnalités du système**
  - **http://www.laszlosystems.com/partners/support/demos/**
- **Pandora**
  - **Une station radio intelligente qui suggère des morceaux de musique en relation avec vos goûts.**
  - **http://www.pandora.com**

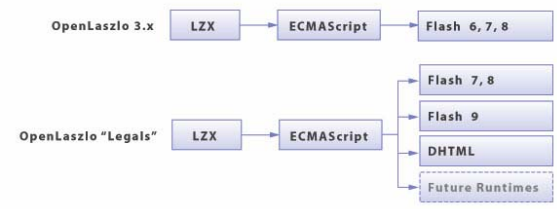
OpenLaszlo

- Comme Ajax
  - XML : chargement asynchrone (XMLHttpRequest)
  - Interface riche
  - Génère du DHTML
  - Utilise JavaScript
- En Plus d'Ajax
  - Liaison déclarative des données
  - Gestion d'États
  - Gestion de Contraintes
  - Lancer de Démons
  - Programmation Objets, Héritage
  - Génère du flash

AJAX et Openlazslo

- Points communs
  - XML et XSLT pour la manipulation et l'échange de données
  - Javascript pour rassembler le tout par programmation
  - Asynchronisme pour le chargement des données
  - Un seul point d'entrée par application
- Différences
  - Ajax : XHTML et CSS pour la presentation
  - Openlazslo : Flash

Le Futur d'Openlaszlo



- Vous pouvez tester Legals (Flash, DHTML)
  - http://www.openlaszlo.org/legals
  - Flash semble +plus rapide au chargement