XIMPEL PLAYLIST VISUAL EDITOR

Student: Javier Quevedo Fernández

Supervisor: Prof, Dr Anton Eliens

VU University of Amsterdam, CS, 2008 - 2009

1. Introduction

XIMPEL is a Framework developed by a group at the CS department of the Vrije University of Amsterdam. In essence, XIMPEL is an eXtensible Interactive Media Player for Entertainment and Learning that offers the possibility to create inter□ active media applications.

XIMPEL applications are described by XML configuration files. These files are created by hand, a task too complicated to be done by a non technical user. For such reason, this project deals with the creation of a Visual Editor to simplify this task so that a wider amount of public can enjoy the XIMPEL eXperience.

2. Editor setup

2.1 End user

In order to use the editor we first have to set it up. This process is rather simple, since it is a flash based application we just have to have any of the available Web browsers, such as Internet Explorer, Firefox or Safari and the Flash Player plugin. Just in case that you are one of the unlucky ones who do not have the Flash Player installed, feel free to download it from http://get.adobe.com/flashplayer/ . Once you have a web browser and the flash plugin installed, simply open the file XimpelEditor.html with the Web browser.

In case that you do not have a copy of the application in your filesystem, you can find it at the following link http://senc.yoteinvoco.com/ximpel/XimpelEditor.html although it is recommended that you grab the latest release from the XIMPEL project Web page since the previous link is no longer maintained.

2.2 Developer

This edit has been implemented in Flex 3.0, using Flex Builder 3 educational edition. Initially, a developer only has to import the source folder into Flex Builder and he/she will already be able to edit, compile or debug the project. The project uses an external library called SpringGraph developed by Mark Shepherd. This library is included in the projects source lib folder, but in case the .swc is missing or a new version has been released, please refer to http://mark-shepherd.com/blog/springgraph-flex-component/ were you can download, get examples or documentation of this fantastic component.

If the developer wishes not to use the Flex Builder tool, he/she can instead use the command line Open Source Flex SDK tools, which are available to download from http://www.adobe.com/products/flex/ . With these tools the developer can also compile, run and debug the project.

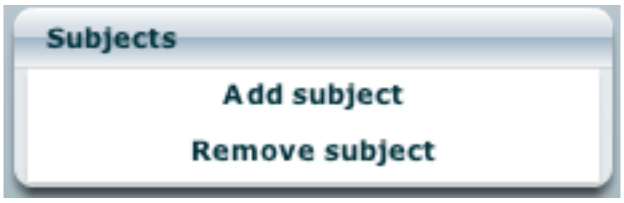
3. Users guide

XIMPEL playlists are organized in subjects with items interlinked between them. Since the purpose of this document is not to explain the architecture of XIMPEL, I recommend the reader to take a look at the DOCS section of http://www.ximpel.net/ in case he or she is not familiarized with XIMPEL in general terms.

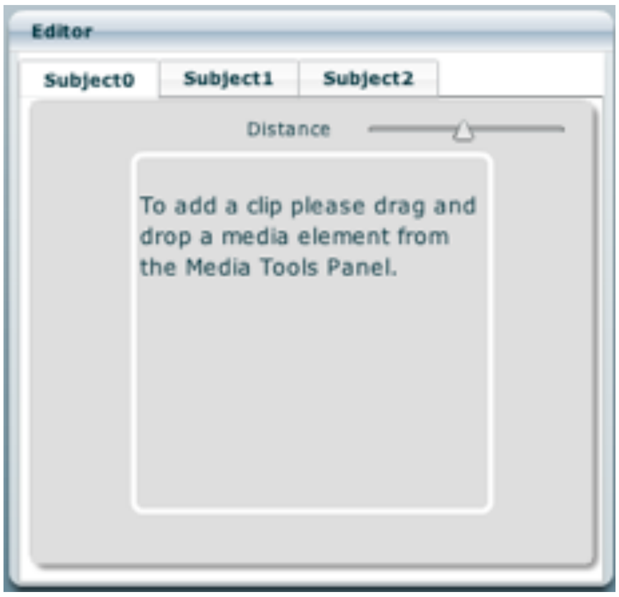
3.1 Add and remove subjects

The first step that we have to perform is to add subjects.

In order to do so, the user has to click the Add subject button.



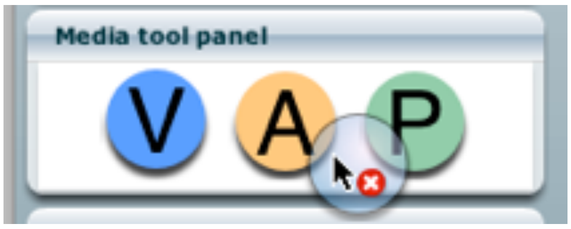
This will add a new tab in the Editor Window with a new subject. We can add or delete as many subjects as we want.



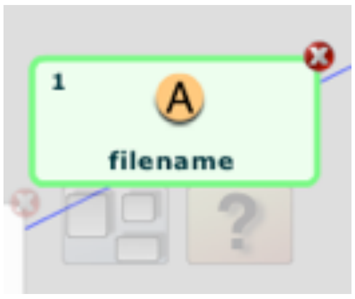
To add a clip please drag and drop a media element from the Media Tools Panel.

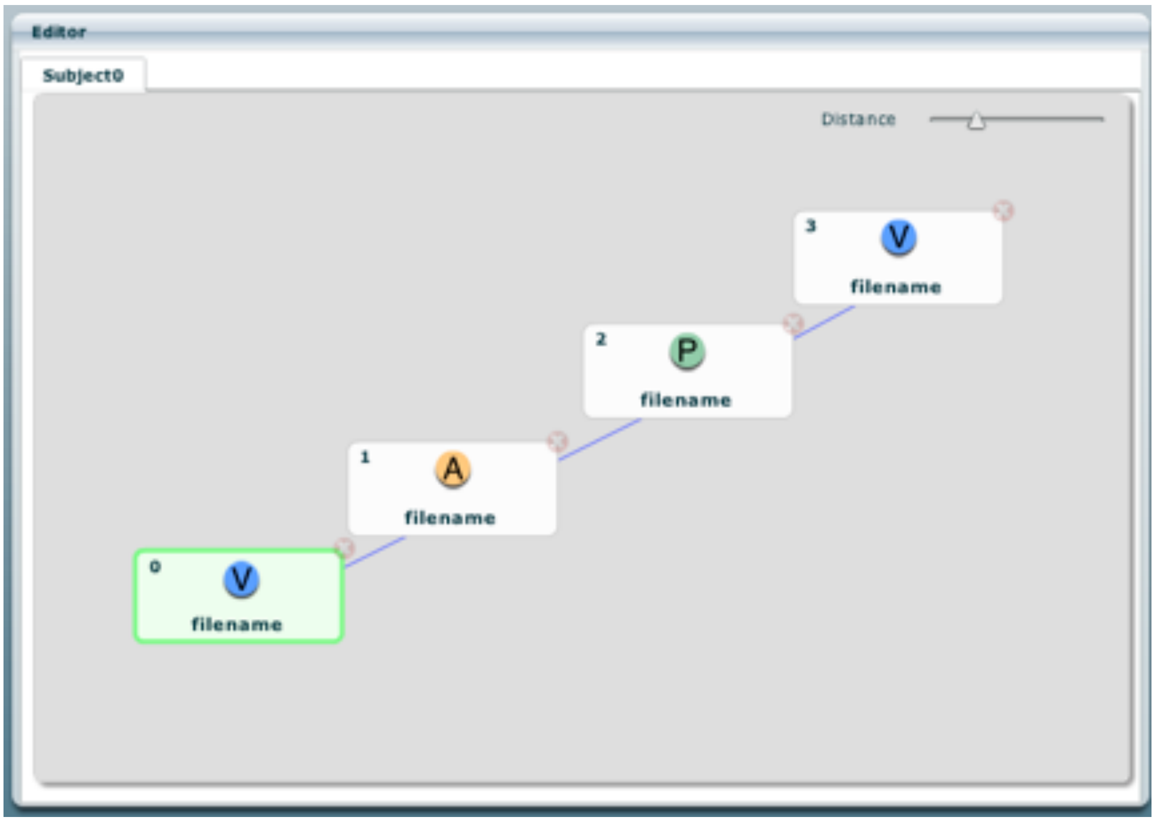
3.2 Add and Remove Media Items

The next step is to Add some Media. In order to do this, we just have to drag and drop an element from the Media Tool Panel into the subject's editor.



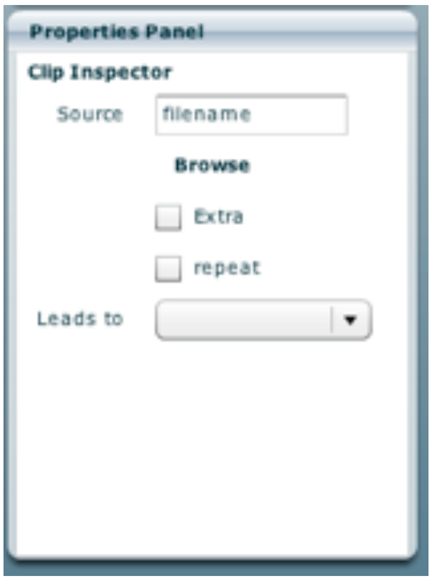
After adding a certain amount of media items, the Subject's editor presents these as a linked graph. We can move it around, add more items which will be appended to the graph or remove items by clicking in the red cross on the top right of the media.



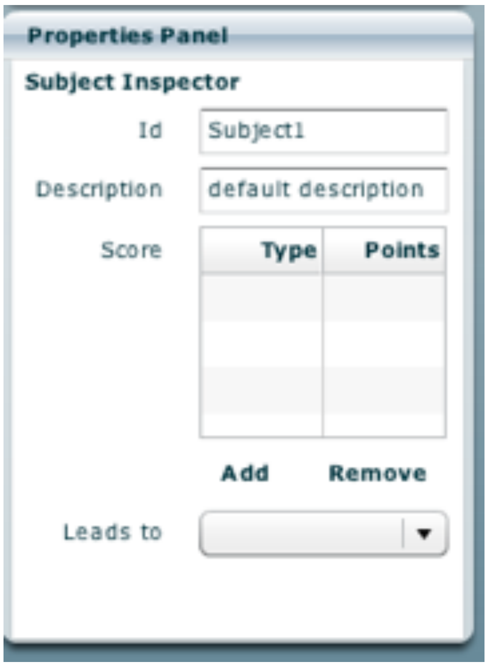


3.3 Configuring the media

Now it is time to setup the media. When we click on a media item, the Properties Panel hosts that items properties, where we modify them. We can browse the filesystem in order to obtain the filename of the media clip which we want it to represent.

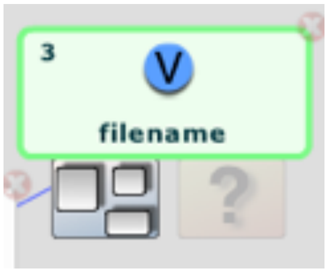


The properties panel can also host the properties of a subject, this happens after we select it.



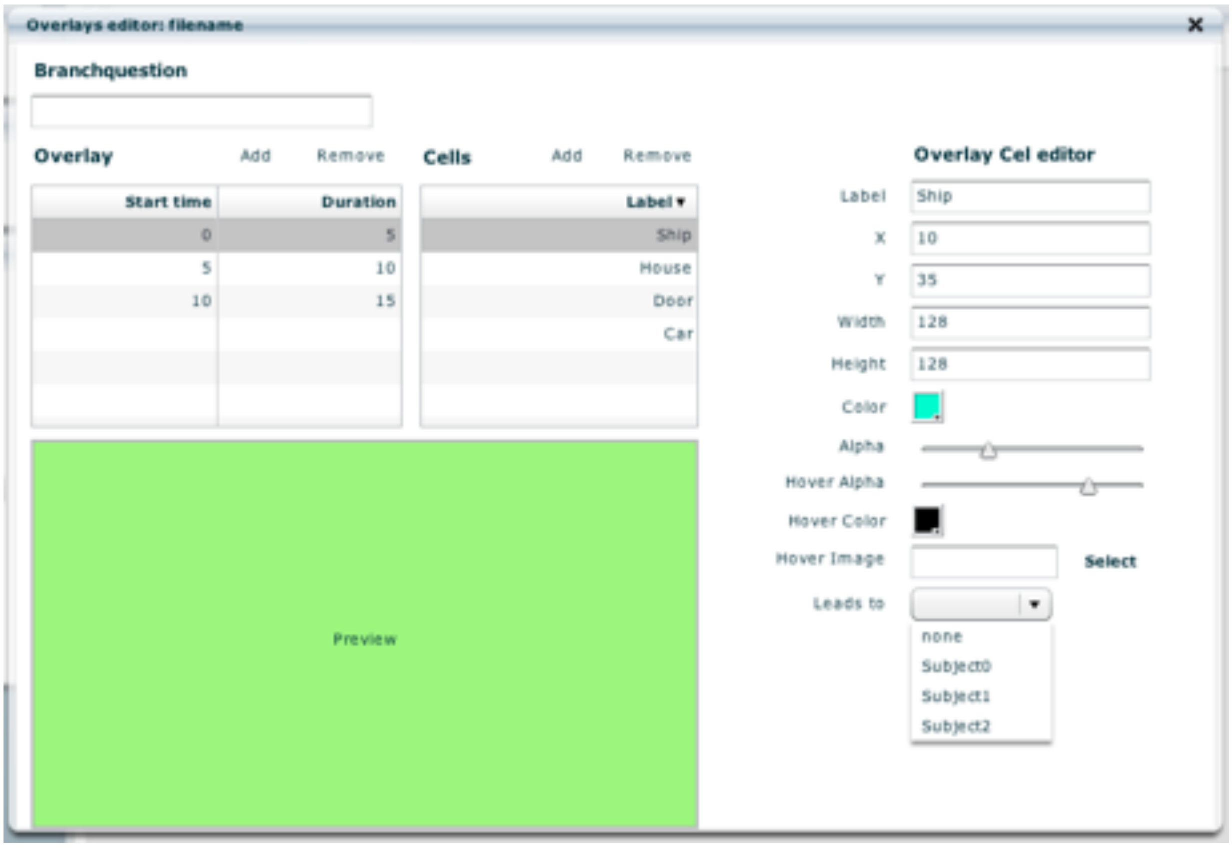
3.4 Overlay and Question editor

The tool includes an overlay editor that we can open by moving the mouse over the item and clicking on the Overlay Editor icon.



With the overlay editor we can add or remove overlays and cells for each one of the overlays.

For each one of the cells we can configure the origin and size for the cell to be displayed, the color, alpha and rest of parameters, plus the subject where the item will transcend to when the overlay-cell is clicked.


Overlays editor: filename

Branchquestion

Overlay Add Remove

Start time	Duration
0	5
5	10
10	15

Cells Add Remove

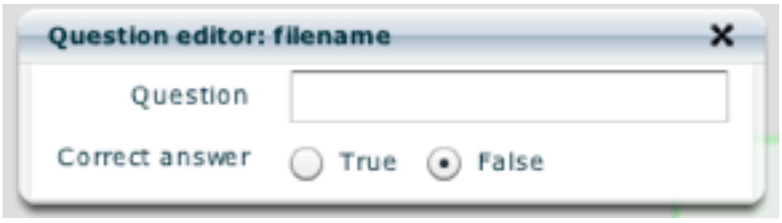
Label ▾
Ship
House
Door
Car

Preview

Overlay Cel editor

Label	Ship
X	10
Y	35
Width	128
Height	128
Color	🟦
Alpha	
Hover Alpha	
Hover Color	⬛
Hover Image	Select
Leads to	none, Subject0, Subject1, Subject2

We can also use the question editor to edit the items question.

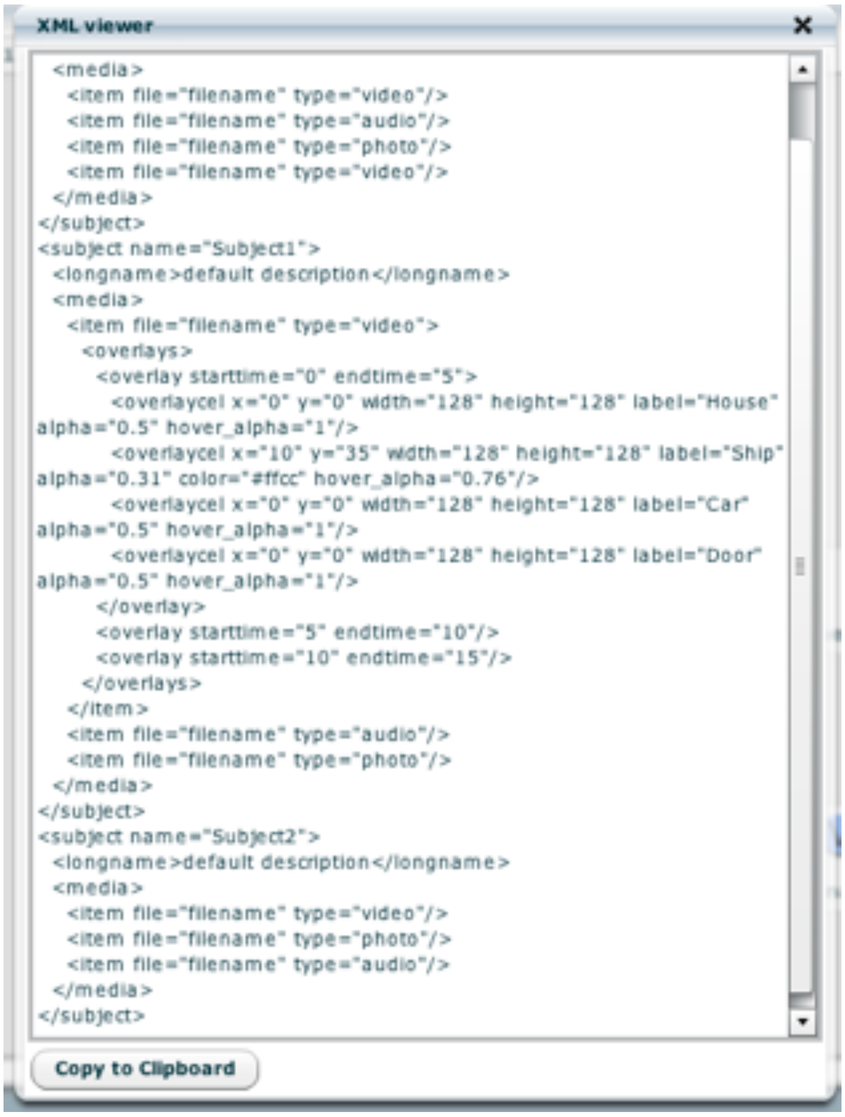


Question

Correct answer: True [✓ False]

3.5 Export to XML

Once we are done editor Interactive Playlist, we can see the XML content by clicking on the top right button with name View XML.

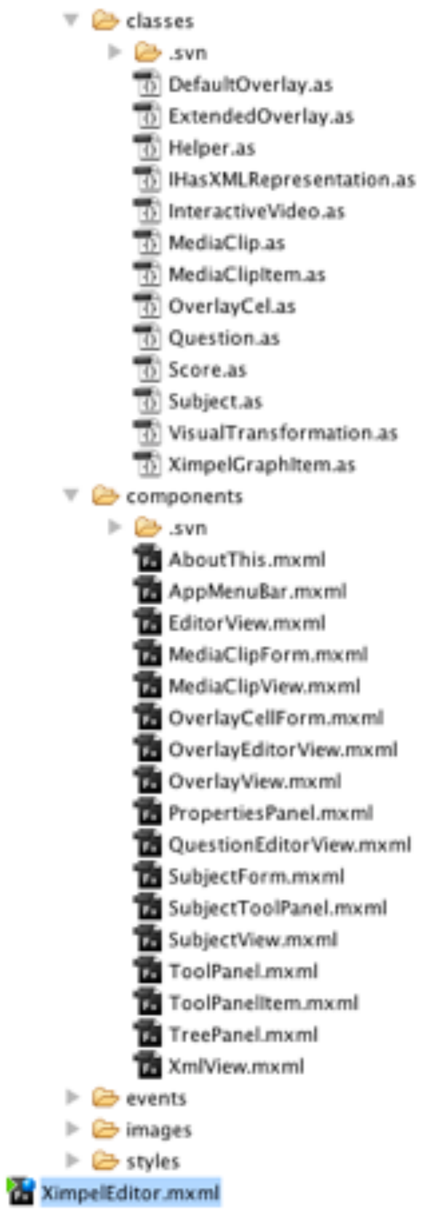


```
  <media>
    <item file="filename" type="video"/>
    <item file="filename" type="audio"/>
    <item file="filename" type="photo"/>
    <item file="filename" type="video"/>
  </media>
</subject>
<subject name="Subject1">
  <longname>default description</longname>
  <media>
    <item file="filename" type="video">
      <overlays>
        <overlay starttime="0" endtime="5">
          <overlaycel x="0" y="0" width="128" height="128" label="House"
alpha="0.5" hover_alpha="1"/>
          <overlaycel x="10" y="35" width="128" height="128" label="Ship"
alpha="0.31" color="#ffcc" hover_alpha="0.76"/>
          <overlaycel x="0" y="0" width="128" height="128" label="Car"
alpha="0.5" hover_alpha="1"/>
          <overlaycel x="0" y="0" width="128" height="128" label="Door"
alpha="0.5" hover_alpha="1"/>
        </overlay>
        <overlay starttime="5" endtime="10"/>
        <overlay starttime="10" endtime="15"/>
      </overlays>
    </item>
    <item file="filename" type="audio"/>
    <item file="filename" type="photo"/>
  </media>
</subject>
<subject name="Subject2">
  <longname>default description</longname>
  <media>
    <item file="filename" type="video"/>
    <item file="filename" type="photo"/>
    <item file="filename" type="audio"/>
  </media>
</subject>
```

If we click on the Copy to Clipboard button on the bottom part of the XML Viewer window, the content of the XML will be copies into the Clipboard of the operating system. We can now open our favorite text editor to Paste the XML and save it into a file, which we can them use as a playlist with the XIMPEL Player.

4. Overall view of the applications design

The application is design following the Model View Controller s (MVC) scheme. There are a set of classes files .as which are the models for the different ele□ ments that we can create or manage. For instance we can find an OverlayCell.as class file, a MediaClip.as class file and on. Additionally, there are several .mxml files which implement the Views and the Controllers, for example the MediaClipView.mxml . Each one of the mxml files instantiates one object of the .as model files, so that every view and controller has one model of is class. All of the class files implement the IHasXMLRepresentation interface, so they all contain a toXML():XML method which is used to export to XML. The different nested elements call each of the siblings toXML() methods. This means that at any time, we can trigger the toXML() InteractivePlaylist method which will build the complete XML tree including all of its children, subchildren and on an on.



```
classes
    .svn
    DefaultOverlay.as
    ExtendedOverlay.as
    Helper.as
    IHasXMLRepresentation.as
    InteractiveVideo.as
    MediaClip.as
    MediaClipItem.as
    OverlayCel.as
    Question.as
    Score.as
    Subject.as
    VisualTransformation.as
    XimpelGraphItem.as
components
    .svn
    AboutThis.mxml
    AppMenuBar.mxml
    EditorView.mxml
    MediaClipForm.mxml
    MediaClipView.mxml
    OverlayCellForm.mxml
    OverlayEditorView.mxml
    OverlayView.mxml
    PropertiesPanel.mxml
    QuestionEditorView.mxml
    SubjectForm.mxml
    SubjectToolPanel.mxml
    SubjectView.mxml
    ToolPanel.mxml
    ToolPanelItem.mxml
    TreePanel.mxml
    XmlView.mxml
events
images
styles
XimpelEditor.mxml
```

The Subject View accepts drops of the different media elements that may exist. When a drag and drop is detected, the Subject creates a new MediaClip View instance, which instantiates a new MediaClip Model object. This new MediaClip is a Child of the Subject where it was dropped and it is linked into the graph as the last element . This way we can add as many subjects and drop as many items as we want into them.

When the overlay editor is created, it is sent the corresponding media item as an instance variable. With this item, it can now add or delete overlays and cells on it.

5. Adapting the XML .

Making modifications of the XML output is very simple. All it needs to be done is to modify the toXML() method of the model classes. For example let's take a look at the MediaClip.as toXML() method:

```
public function toXML():XML
        {
                var attributes:String = "name=\"" + id + "\"";
                if(leadsTo && leadsTo != "none"){
                        attributes = attributes + " leadsto=\"" + leadsTo + "\"";
                }
                var subject:XML =
                <subject {attributes}>
                        <longname>{description}</longname>
                </subject>;

                for (var j:int = 0; j < scores.length; j++) {
                                subject.appendChild(Score(scores[j]).toXML());
                }

                var videosXML:XML =
                <media></media>;

                for (var i:int = 0; i < media.length; i++) {
                        if(media[i] is MediaClip){
                                videosXML.appendChild(media[i].toXML());
                        }
                }
                subject.appendChild(videosXML);

                return subject;
        }
```

As we can see , the toXML() function first builds an string with the different at tributes, after that, it creates the XML node and loops around each one of the siblings adding the XML output of each of them. Modifying this function will result in modifying the final XML output.

6.Creating new Media Items .

The project includes a folder called config with the configuration file toolpanelitems.xml.

This is read when the application starts and it creates the corresponding tool items that are described in the file. In order to add new custom media items we have to add a new entry in the file with the appropriate parameters.

```
?xml version="1.0" encoding="utf-8"?>
<toolpanelitems>
      <toolpanelitem type="video" name="Video clip" description="Add a new 
video" tooltiptext="Add a new video" 
toolpanelicon="net/ximpel/images/videoIcon.png" 
viewicon="net/ximpel/images/videoIcon.png" />
      <toolpanelitem type="audio" name="Video clip" description="Add a new 
audio" tooltiptext="Add a new audio" 
toolpanelicon="net/ximpel/images/audioIcon.png" 
viewicon="net/ximpel/images/audioIcon.png" />
      <toolpanelitem type="photo" name="Video clip" description="Add a new pic-
ture" tooltiptext="Add a new picture" 
toolpanelicon="net/ximpel/images/pictureIcon.png" 
viewicon="net/ximpel/images/pictureIcon.png" />
</toolpanelitems>
```

Each toolpanelitem node includes the type of media, which will result in that custom media type, the description, name, the tooltip text, the icon for the panel and the icon for the media clip view.

7. Coding tricks.

The editor does not implement many strange coding tricks or hacks. New developers should be-aware that not all of the property panel editors are implemented the same way. The subject property panel for instance does not use data bindings, instead it handles the commit events of the textinputs and then it modifies the values of the model object. This does not happen in the media clips for example. The reason for this is that at the time I was implementing the Subject Property Panel I did not know how to take advantage of Flex databindings and by the time I actually did, it was too late to change it.

Another trick that the developer and user should be aware of is that the LeadsTo attribute in some cases contains a string and in some cases an object. This means for example, that if you set the leadsTo attribute of a subject it will maintain only the string of the subjects targets name, therefore if after setting it, the targets subjects name is changed, the original subject will not detect the changes. This does not happen in the case of the MediaClips, those are implemented in a way that the leadsTo attribute links to the object that it really leads to, and when the XML output is produced, it uses the objects name, therefore the result is always correct no matter what changes you apply to the targets object after you have selected the source leadsTo attribute.

Thanks to Anton Eliens and Winoe Bhikharie