Increasing Widgets Interoperability at the Portal Level

Stéphane Sire

EPFL IC IIF GR-VA
BC 156 Station 14
CH-1015 Lausanne
stephane.sire@epfl.ch

Alain Vagner

CRP Henri Tudor
29, Avenue John F.Kennedy
L-1855 Luxembourg-Kirchberg
alain.vagner@tudor.lu

Abstract. We make the statement that widget APIs can be extended to create Web portals where simple interactions between widgets are possible.

Introduction

Current Web portals and portlet architectures allow user-defined placement of widgets on a multi-columns layout and the association of user preferences with each widget. This is quite convenient in a PLE to realize the visual integration of several services through the provision of a widget for each service. However, this is not enough to qualify the Portal as a mashup application, as far as the absence of communication between widgets does not allow interaction between services to be triggered from the composition portal.
In order to design a cross-widget communication mechanism to allow interaction between services at the portal level, we propose to start first with some simple scenarios. Then we discuss the requirements for a solution and we propose an extension of a widget API compatible with the Widgets 1.0 working draft of the W3C and that is being developed in the framework of the PALETTE project.

Scenarios of Cross-Widget Communication

The choice of the initiator of cross-widget communication distinguishes two types of scenarios. In the first choice, the user has the direct initiative though a drag and drop operation. In the second choice, a widget has the initiative through a change in its state, which can be triggered as a consequence a system action, or by the user.

Drag and drop Scenario

The drag and drop scenario is an extension of the drag and drop operation which is natural in direct manipulation user interfaces such as operating system desktops.

In the PALETTE project for instance, a common repository service hosts documents created by a community of practice. A search widget allows to query these documents and to present them in a result list. Some of these documents are template documents which are used in an editing service to author structured documents. In that case a convenient use of drag and drop would be to drop a template document from the search widget to the editing widget, to start the authoring of a new document from this template. In addition, the editing widget can show a preview of the template.
Another example is a contact widget displaying the list of contacts of the user with eventually some online status information. Such a list is a shortcut to select a user which can then be used to perform all kinds of operation into other services. For instance, if a widget is available for negotiating meeting arrangements, it may be convenient to drag and drop the users to invite to a meeting directly from the contact widget to the meeting calendar widget.

Widget State Coupling Scenario

When a news widget receives a news item about a particular company, it could tell a stock quotes widget to display any changes in that company's share price (requirement 33 of W3C Widgets 1.0 Requirements working draft). This can be generalized to any widget that holds some data which can be updated. Updates may come from the system, or they can be triggered by user actions, such as when a user selects a buddy in her contact widget. In both cases it may be interesting to automatically propagate widget state change to any other widgets that can set their state accordingly.

Raised Issues and Requirements

The scenarios raise issues at the user level and at the technical level. At both level it is worth to search for a generic solution that would be implemented once into a portal and its widget APIs.
At the user's level, drag and drop scenarios require that users be able to detect when a drag operation is permitted from a source widget and to identify the target widgets. The first feedback is usually provided through a cursor change when flying over a drag source, and with an highlighting when flying over a potential target. This allows the provision of a generic mechanism that would work at the portal level.
State coupling scenarios require that users must be able to anticipate which widgets influence each others when they are instantiated together on the same portal. Here the designer faces with two solutions: either to let users manage explicitly the coupling between widget states, or to handle it automatically. The second case seems the easiest from a user's point of view. However, for some cases when the users wouldn't like to allow a coupling between two widgets states, it seems necessary to provide a portal with a mechanism to “decouple” two widgets, and to “undo” that decoupling. It also seems necessary to highlight the couplings that follow a widget instantiation.
At the technical level, drag and drop scenarios require a way to detect the content of a widget that can be dragged. They also require to define what data is transmitted

during the drop operation and a way to bind the transmission of data with some actions on the target widget.
State coupling scenarios require a way to describe the state of a widget and a way to bind a state update with some actions on a target widget. They also require to define a way to copy, or to access, state data of the initiating widget from bound widgets. According to the user level requirement of automatically coupling widget states, it is also required to define a protocol for matching widgets states together.

Proposition of a declarative solution

We suggest a declarative API in order to simplify the deployment of drag and drop and widget state coupling scenarios. We propose to extend the XML configuration file that comes with most widget APIs specifications, so that it can describe drag and drop bindings and state coupling bindings between widgets.

Drag and drop

The draggable content and the transmitted data are defined at the same time with a new “dragref” attribute that can be added to the presentation language (XHTML or SVG for instance) with a special namespace prefixed “w:”. Thus, the following example declares a list item which can be dragged :

```
<li w:dragref="template?http://repository.org/aFile">
Some name</li>
```

The portal implementation uses the “dragref” attribute to generate on the fly a class attribute value that allows to define a visual alteration (such as cursor change) of the drag data island. In the future it will be possible to do it directly with CSS 3.0 namespace compatible selectors.
We propose to use a convention to declare the drag data as a combination of a keyword followed by an interrogation point followed by a URI as in the example above. The idea is that if the URI does not contain meaningful data, it can point to a Web service that would retrieve the data.
Finally, to declare bindings between some drag data and a widget as a drop target as a whole, we propose to extend the XML configuration files of the widget with a “binding” and a “submission” elements. These elements match the “dragref” contents with regular expressions. This is illustrated below. With such as declaration, the “getContact” Javascript method of the widget will be called when drag data matching the “src” attribute is dropped onto it :

```
<w:binding src="contact\?(http://id.mycop.org/.*)">
          <w:submission action="javascript:getContact"/>
</w:binding>
```

The drag data is the URI that allows to retrieve contact details from the community identity service. The Javascript method is called with the matching sub-expressions of the regular expression of the binding as parameters.

Widget state coupling

We are still investigating a syntax to declare widget state coupling in a declarative form directly within the widget configuration file.
The concepts are well defined. First, each widget must declare it's state model. This could be done in a way similar to the XForms instance element. Second, each widget must declare its potential bindings with other widgets. A syntax similar to the drag and drop binding declaration above is possible, the only difference would be that instead of matching “dragref” content with regular expressions, it could be possible to use XPath expressions pointing to other widget instance element models as with XForms bindings. Finally the third step would be to describe an action to perform for each binding. Going a step further it could eventually be possible to directly link parts of the target widget instance model with corresponding parts of source widgets instance model, or to define a transformation between both, such as an XSLT one.

Related Work

Several efforts are already under way to introduce cross-widget communication into widget APIs such as into the W3C Widgets 1.0 working draft, or the Java portlet specification. However these specifications are still in development, and they do not address the user interaction issues contrary to our proposition.
The integration of services at the presentation tier is also an emerging topic, with some authors suggesting some extensions of the SOA architecture to the client side for the integration of the presentation components, while some Javascript frameworks are also appearing proposing different styles of Javascript dataflow architecture or publish-subscribe patterns to connect widgets together.

Conclusion and Future Work

The extension of Web portals toward a full Web desktop metaphor with drag and drop is intuitive. It can also be seen as a “manual” mashup, where users mix and match data coming from and going to different services with explicit actions. Actually we propose to declare the bindings into the configuration file of the widgets. This is a static solution in the sense that the bindings must be defined at the time the widget is embedded into the portal. A potentially more dynamical solution would be to define the bindings as a part of widget preferences. That would allow users to create their own bindings, or to dynamically imports new bindings as they are available.

Acknowledgements

This work is supported by the IST programme of the European Commission (DG Information Society and Media) through the PALETTE Integrated Project.