A Messaging API for Inter-Widgets Communication

Stéphane Sire
Micaël Paquier
EPFL IC IIF GR-VA
BC 156, Station 14
CH – 1015 Lausanne
+41 21 693 12 18
{firstname.lastname}@epfl.ch

Alain Vagner
Jérôme Bogaerts
CRP Henri Tudor
29, Avenue John F. Kennedy
L-1855 Luxembourg-Kirchberg
+352 42 59 91 - 1
{firstname.lastname}@tudor.lu

ABSTRACT
Widget containers are used everywhere on the Web, for instance as customizable start pages to Web desktops. In this poster, we describe the extension of a widget container with an inter-widgets communication layer, as well as the subsequent application programming interfaces (APIs) added to the Widget object to support this feature. We present the benefits of a drag and drop facility within widgets and conclude by a call for standardization of inter-widgets communication on the Web.

Categories and Subject Descriptors
H.5.3 [**Information interfaces and presentation**]: Group and Organization Interfaces – *synchronous interaction, web-based interaction*.

General Terms
Design, Experimentation, Human Factors, Standardization.

Keywords
Widget, Portal, Start page, Drag and Drop, Communication.

1. INTRODUCTION
iGoogle or Netvibes are some of the most used customizable start pages integrating individual Web widgets. In this paper, we term the above applications as *widget containers*. They allow users to gather miscellaneous pieces of information within a restricted area, aiming at being able to rapidly visualize and use them. This idea is derived from window managers in operating systems desktops. However, current widget containers provide neither interaction nor communication features among their widgets.

Nevertheless, benefits of inter-widgets communication are multiples: ability to redirect user's inputs to several widgets, intuitive reuse of complex data structures, etc. We claim that different services accessible through widgets could be combined, resulting in the simplification of often repeated actions. The drag and drop technique between widgets is probably the most intuitive feature that can be carried out to facilitate widgets interaction.

A few concrete examples illustrate this: drag a city item from a geographical map and drop it to a weather widget, drag an English RSS feed entry and drop it to a translation widget to read the full article, etc.

WWW 2009, April 20–24, 2009, Madrid, Spain.
ACM 978-1-60558-487-4/09/04.

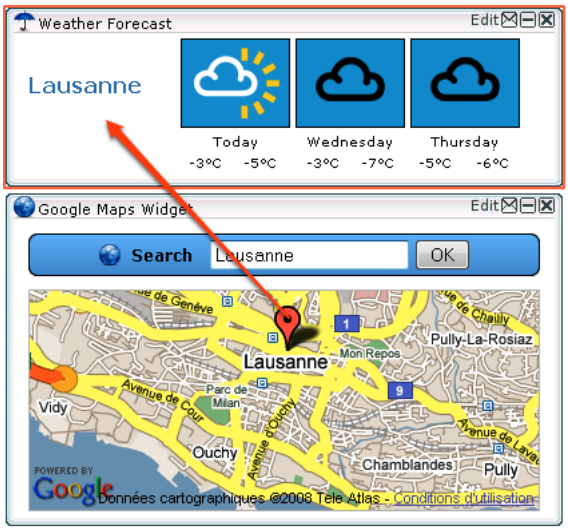

Figure 1. Drag and drop a landmark to get the corresponding weather forecast.

In the frame of the EU FP6 PALETTE Project [5], we have developed a widget portal that allows users to interact with several services, especially in using drag and drop operations between widgets.

The portal, called *myWiWall* [4], is available with an open source license and is based on the W3C widget 1.0 specification [1]. The next sections describe the two different APIs that we have added to the Widget object: inter-widgets messaging and drag and drop messaging. We also give an overview of the implementation before the conclusion.

2. INTER-WIDGETS MESSAGING API
The inter-widgets messaging API follows an event-based communication model: a source widget can communicate with one or multiple target widgets by using event messages. Target widgets manage their subscription to events with the *addWidgetEventListener* and *removeEventListener* methods. Source widgets send an event with a *fireWidgetEvent* method.

Events can be fired using the following modes: Unicast (unique receiver), Multicast (multiple receivers) and Broadcast (all instantiated widgets being receivers). The mode is specified as a list of accepted sources (indicated by their widget identifier) on the event subscription method, and by a list of targets on the event firing method. A null value implies any source or any receiver.

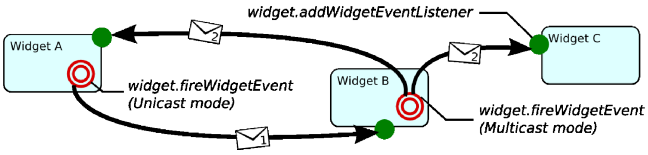

Figure 2. Global overview of the inter-widgets messaging API.

These different modes allow widget developers to manage different kinds of communications.

Event messages transmitted to widgets are characterized by an event type and some associated data. In our implementation the event type can be any string. However, in practice we recommend to define an event type as a URI so that event semantics can be described with semantic Web techniques. This is also a good way to setup common vocabularies of events shared by widgets.

Two strategies may be used to populate the transmitted data. The first one consists in transmitting any object to the target widget. The second one consists in passing a URI string in order to let the receiver de-reference it and trigger the appropriate behavior. We rather selected the first strategy, because it gives a direct access to the data and it also can emulate the second strategy.

3. DRAG AND DROP MESSAGING API

The drag and drop messaging API is layered on top of the inter-widgets messaging API. Drag and drop events are like other events: they are defined by an event type and some associated data. This way subscription to drop events also uses the *addWidgetEventListener* method. In addition target widgets must declare their interest in a drop event with the *bindWidgetToDropType* method that takes a given event type as parameter. This is necessary to know if a widget can receive a drop event during a drag action and to display proper feedback around the widget frame. Currently there is no way to define multiple drop zones into a widget; this should be addressed in the future. The DOM element of a widget tree that can be dragged onto other widgets must call the *addDragData* method. It defines the associated event type and the event data that will be transmitted to the target widget. It also contains an extra string argument that defines a tooltip message to display during the drag action.

4. IMPLEMENTATION

The *myWiWall* application is made of two parts: a widget engine, written in PHP code with mySQL database storage, and a widget container which is a client-side library written in JavaScript.

The messaging and drag and drop APIs have been implemented entirely client-side, as extensions to the JavaScript library. They are made of two parts: an inter-widgets communication layer, and a drag manager object. It detects drag events and displays the proper feedback and affordances to the end user using a div layer placed on top of the screen with a high z-Index. When the user drops an object, the drag manager creates an event and transmits it to the target widget using the Unicast mode of the inter-widgets messaging API.

We detect when the mouse pointer of a drag action enters a widget frame with a custom mouse event peeking algorithm based on the frame position of each widget and on the current mouse position.

5. RELATED WORK

The Java Portlet specification (JSR 286) [2] and the Cross-document Messaging API described in the W3C HTML 5 draft recommendation [3] inspired the design and development of the event-based messaging API of *myWiWall*.

Desktop widget engines such as Apple Dashboard or Microsoft Windows Sidebar enable widget developers to create widgets that are able to receive data from a drag and drop operation that began on the operating system's desktop-side, but do not provide natively *widget to widget* drag and drop operations. Noteworthy widgets APIs such as Netvibes UWA or Google Gadgets API do not yet implement unified methods for inter-widgets messaging and drag and drop handling.

6. FUTURE WORK

We have started to discuss some declarative versions of these APIs. They could be used for instance to declare subscriptions to events directly from a widget manifest file, and to declare drag events directly from the XHTML source code.

We have also started to discuss some extensions of the messaging APIs towards distributed messaging. This way, a widget generated event could be received by multiple users viewing the same widget. This could be used for instance to easily implement synchronous image sharing or chat widgets. The distribution of events would require the introduction of COMET-based communication protocols into the widget engine.

7. CONCLUSION

The inter-widgets and the drag and drop APIs only require the addition of 5 methods to the Widget object. They are easily implementable into a small footprint JavaScript library and might be adopted among a wide range of existing widget portals. This work is still in progress, it is being discussed on a public mailing list where everyone is welcome [6]. We are planning to submit it to the W3C as a Member submission as we feel it is important to agree on a common API to increase interoperability between widgets on the World Wide Web.

8. ACKNOWLEDGMENTS

PALETTE is an Integrated Project supported by the IST programme of the European Commission (DG Information Society and Media, no. 028038).

9. REFERENCES

[1] Bersvendsen A., and Caceres M. (eds.). Widgets 1.0: APIs and Events. http://dev.w3.org/2006/waf/widgets-apis/

[2] Hepper S. JSR 286, Portlet Specification 2.0. http://www.jcp.org/en/jsr/detail?id=286

[3] Hickson I., and Hyatt D (eds.). HTML 5 W3C working draft. http://www.w3.org/TR/html5

[4] myWiWall portal. http://code.google.com/p/mywiwall/

[5] PALETTE project. http://palette.ercim.org

[6] Talk about Widgets. http://tinyurl.com/and5k5