Usability and Inter-widget Communication in PLEs

Erik Isaksson and Matthias Palmér

Uppsala University

Abstract. The purpose of introducing inter-widget communication is ultimately to offer a better user experience. Possible advantages include allowing a smoother flow of activities across widgets, increasing the amount of possible activities, and strengthening widget quality due to widget development becoming more specialized when common tasks can be delegated. However, widget communication also risks worsening the user experience, since it may cause the user interface to become more complicated and unintuitive. We introduce a new approach termed Open Application, where we let usability concerns guide our design.
The main points of Open Application's inter-widget communication are that all significant actions in a widget should be broadcast to all other widgets in the browser, and that a receiving widget comprehending enough of an event may react directly or notify the user about further actions that may be taken. A conceptual PLE user interface is also introduced, followed by an example.

Keywords: widget communication, partial semantic interoperability, personal learning environment, usability

1 Introduction

Communication between widgets in a Personal Learning Environment offers the ability of combining the services of several widgets. A user action in one widget can result in responses in other widgets. Based on that action, those widgets can then propose further actions. Whereas widgets without such communication will be unresponsive to what occurs in other widgets, communicating widgets will be responsive to events in their context. In this way, a widget will have the potential to become more useful as other widgets are added (a kind of widget "network effect").

Previous inter-widget communication frameworks have relied on users or developers manually connecting widgets with each other in order to enable communication [4]. For users, this has been proven confusing, showing the need for a solution that "just works", automatically, for any widgets that the user chooses to use. Relying on developers connecting widgets is limiting, since it is generally not possible to know in advance all of the widgets that will be available to users.

The aim of this work is to improve the usability of inter-widget communication. The solution is in part a framework that offers support for the communication that is needed, and in part an experimental user interface that makes use of the framework. It should be noted that the communication is layered on top of an existing browser-provided communication channel.

The communication applies only to one browser. I.e., the communication is between widgets running in the same web browser instance. There are, however, other types of widget communication. We suggest the classification in table 1.

Table 1. Widget communication matrix.

1: Same-browser inter-widget comm.	2: Same-browser intra-widget comm.
3: Cross-browser inter-widget comm.	4: Cross-browser intra-widget comm.

As an opposing example to inter-widget communication, the communication in Google Wave communication is intra-widget and cross-browser. We intend to combine such cross-browser communication with our inter-widget communication (orthogonal to each other) at a later stage.

There are many other systems that enable similar inter-widget communication (see [1] where Open Application is also included). However, we have not found any other system that features partial semantic interoperability.

2 Usability Concerns

Our aim with introducing a new way to do inter-widget communication is firstly based on user needs, and only secondly on technical elegance. In this section, we list three general usability concerns related to widget-to-widget communication:

Just works: There should be no configuration step that involves wiring of widgets together in preparation of inter-widget communication.

Automatic: Communication should be initiated between widgets as a byproduct of well-integrated actions, for instance, other widgets should be notified directly when a selection is made in a list rather than requiring pressing a "Send" button. The communication should be directed to all widgets within a given context for potential further actions.

User in control: No communication between widgets should be initiated without a user performing a relevant action. Widgets receiving the communication should not react in a way that has side effects without involving the user. Visual cues should lead the user to widgets that provide the option to do something further with the sent information. This and the previous concern need to be balanced.

3 Technical Overview

As part of an initiative called Open Application, we have developed an event-based inter-widget communication framework. While initially intended for communication between widgets based on OpenSocial [5], it can be used also in other environments (e.g., other widget types). The following briefly covers the principal choices that have been made in the development of the framework.

3.1 Events

As widgets are to be responsive to what occurs in other widgets, it is natural that the point in time for a widget to send a message is when something has occurred, that other widgets should know about. This is in contrast to Google Wave, where messages synchronize the internal state of a single shared widget instance.

A widget will thus send a message in the form of an event when the user performs a non-negligible action [7]. Such an event could be, for instance, that the user has made a selection. This will then be communicated as a select event. Information such as the event type (see table 2) is included as metadata concerning the message itself.

Table 2. Event types.

Event	Description
state	The widget has changed its state in a way that does not directly reflect any particular resource.
modify	The given resource has been modified.
select	The given resource has now been selected.
unselect	The given resource has now been removed from the selection.

3.2 Event Distribution

Events can be distributed among widgets in different manners. The following three approaches have been considered:

Direct widget subscription: Widget A subscribes to widget B to receive any events from widget B. Understanding the event data is not a problem, since widget developers will need to have knowledge of every subscription and can solve it in a case by case manner. The negative side of this is that all widgets communication will require active planning from developers. When the amount of widgets grows the communication step will require more and more effort, i.e., endless updates. The only way to achieve more flexibility in this case is to let users wire widgets together, but this contradicts the “just works” requirement.

Topic subscription: Widgets interested in a specific topic subscribes to an event bus dedicated for that topic. All events sent on one event bus are more or less required to follow some agreement on data structure and corresponding semantics. Hence, the widget developer will make the widget understand the events, when he decides to make the widget subscribe to a certain event bus. In principle, this could make widget communication something that "just works" and not require endless updates from the developers to make it communicate with all other widgets. However, the risk is that widget developers do not quickly agree on a set of well known topics and accompanied data expressions. An extreme case would be one event bus per widget, leading to a situation which is comparable to direct widget subscription, with endless updates as a result. This approach also introduces the risk of evil widgets breaking the users expectations of confidentiality.

Broadcast: Every widget broadcasts everything that could be of interest to all other widgets. This approach will not require widget developers to do endless updates to just make the events reach relevant other widgets. However, understanding event data (semantic interoperability) is now a bigger problem since it is not restricted by originating widget or topic. (If we base understanding of event data on the originating widget, we have effectively returned the direct widget subscription case, forcing widget developers to make endless updates. This time it is not to receive the event, but to understand it.) The risk of evil widgets is the same as for topic subscription.

In the Open Application framework we have decided to use the broadcast approach and try to solve the semantic interoperability problem (see the later discussion on partial semantic interoperability).

3.3 Sharing and Security

The usability concern termed "user in control" has many consequences. How do we keep the user in control, when widgets automatically broadcast the user's actions as events? While we have not found any feasible manner, in which we could enforce restrictions programmatically, we will introduce three sharing levels. These levels restrict how event data, or data partly derived from event data, may be stored or communicated. We suggest that developers make their widgets obey these sharing levels. That widgets actually do so may be verified upon inclusion of widgets into a widget store, through a manual review process.

Level 1. Session: Receiving widgets may only temporarily react to the event. The data is not allowed to be stored beyond the current session or for that matter leave the client. Hence, if the client is restarted, there should be no trace of the event in the receiving widgets or in any third-party service.

Level 2. Client: Receiving widgets may react permanently to the event, but the data may still not leave the client. Hence, widgets may store data (in part derived from the event) in a permanent cookie or in offline storage, so that the receiving widgets apearance and behavior may still be affected after a restart of the client.

Level 3. Service: Receiving widgets may react permanently as well as store or communicate data with external services.

3.4 Partial Semantic Interoperability

From the usability requirement of “automatic”, we think it is important that arbitrary widgets can communicate with each other, even when the developers have not explicitly prepared for this combination. However, it is not enough to receive the information, for widgets to be said to communicate, they must be able to interpret the information which they receive, in accordance with the intentions of the sender. This is called *semantic interoperability.*

In the Open Application framework, we only require that widgets understand parts of the information sent. The alternative would be to either limit what can be communicated or require an herculean effort by developers to continuously update every widget, so that it understands every other widget. Thus, every widget developer should make his widget understand as much as possible that is relevant for the operations it supports. This is called *partial semantic interoperability.*

To lay the groundwork for partial semantic interoperability, messages are crafted with a very flexible, resource and property-centric approach. A message consists of the description of a resource, which is the subject about which the message is concerned. The description is inspired by metadata standards such as Resource Description Framework (RDF) [2] with the Dublin Core metadata element set [3] as a basic vocabulary (a generic set of properties). There is no limit on what metadata vocabulary a widget may use, but it is encouraged that widgets reuse the same vocabulary, whenever possible, and that they understand other vocabularies when several are in common use. Then, for a widget, partial semantic interoperability means looking into every message and only accepting it when the widget recognizes enough properties that are useful input to further interaction in the widget.

3.5 Syntax

The events are communicated as messages wrapped in envelopes. The message contains metadata about the subject of the event, e.g., the document having been selected, while control data needed for the communication, or meta-metadata, will be kept directly in the envelope.

The whole envelope including the message must be valid JSON. The parameters within the envelope must be valid according to the Open Application specification, and the message must be in one of several approved formats (see table 3).

3.6 Channels

For communication to take place, there must exist a channel through which it can take place. This is not a trivial problem for the inter-widget communication. Depending on the environment (the widget container and the web browser), various channels may or may not be available. Also, the channels may offer different features. ([8] includes a discussion about possible channels.)

Table 3. Message formats.

Format	Description
namespaced-properties	Predicate-object pairs in a JavaScript object: a simple way of declaring properties for the resource relevant to the event.
rdf-json	RDF as JSON: when full RDF is required to describe the resource relevant to the event.

The HTML5 standard currently being developed provides a facility for message passing: postMessage [6]. This is the preferred channel, as it will work in containers even without explicit inter-widget communication support (e.g., iGoogle), is compatible with same-origin security limitations, and contains browser-provided security features. It is also widely supported in current web browsers. When this facility is unavailable (such as in too old web browsers), an OpenSocial feature called gadgets.pubsub is attempted instead (which is, however, functional only in some OpenSocial containers).

3.7 Library

The framework is implemented as a JavaScript library that needs to be included by widgets making use of the inter-widget communication. The script is available on the Open Application project website [9].

4 Conceptual User Interface

This section describes a conceptual user interface (figure 1), that supports inter-widget communication and takes the usability concerns into account. Note that the features of the conceptual user interface are not a necessary consequence of relying on the Open Application inter-widget communication. However, we regard a well-thought-out UI to be an integral part in meeting the usability concerns, and as such we propose the following features to be implemented when applicable.

4.1 Widget Features

As it is up to each widget to implement the user interface for inter-widget communication, it is first of all important that widget developers try to be consistent in their design.

Updates: Messages about events are to be sent automatically when the events occur, without requiring the user to, e.g., click on a “Send” button. Widgets are then to be responsive to events in their context by updating themselves to reflect the user’s activities (when there is no adverse side-effect that would

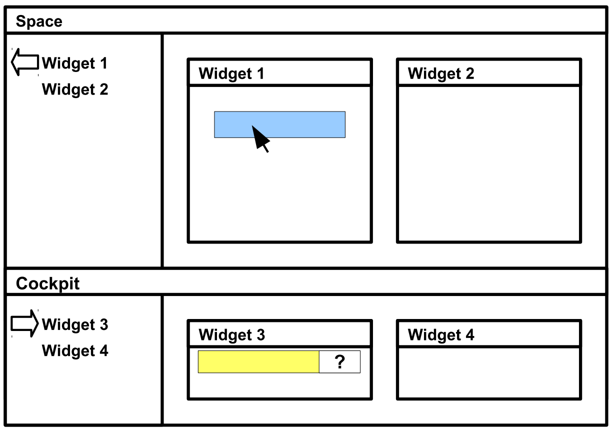


Fig. 1. Conceptual user interface of a PLE with inter-widget communication, displaying two widgets in a space and another two widgets in a cockpit, where a widget in the space is broadcasting an event that is accepted by a widget in the cockpit.

leave the user feeling out of control). One type of update could be a map widget moving the current view, in order to show the location of a relevant resource.

Affordances: Upon receipt of an event, actionable elements (e.g., buttons) are provided for indicating actions that can be performed (which may result in new, second order events). The user's attention needs to be attracted without distracting him or her.

4.2 Container Features

PLEs, whether they are widget containers specialized for learning or more general containers (such as iGoogle), can provide features to make the communication more accessible, and also themselves communicate with contained widgets.

Spaces: Spaces offer a context within which widgets communicate. A PLE may choose to limit communication with widgets outside of the space.

Cockpit: The cockpit contains personal widgets, that are always available to the user, and that are able to communicate with the widgets in the space which the user is currently working on.

Indicators: The indicators give a visual hint to show that inter-widget communication is taking place, in addition to hints already displayed by the widgets themselves.

5 Example

As an example scenario, we have a user who is reading news stories (e.g., as part of learning a foreign language). The user finds news stories in a widget that displays an RSS feed. If a story is of such interest that the user wants to keep a reference to it, it can be added to the user's portfolio, by means of a portfolio widget. This allows the user to access the news story even after it has disappeared from the news feed. Finally, news stories, whose metadata include map coordinates, will be displayed with markers in a map widget.

5.1 Selecting a News Story

When a news story is selected, the following event is sent out:

```
{
  event: "select",
  type: "namespaced-properties",
  uri: "http://www.example.com/news0001'',
  message: {
    "http://purl.org/dc/elements/1.1/title": "News story",
    "http://simile.mit.edu/2005/05/ontologies/location#coordinates",
"64.10,-051.45"
  }
}
```

5.2 Adding to Portfolio

The portfolio widget receives the above event, with some additional security-related properties added on the way. Since the portfolio widget recognizes a resource that may be added (there is a URI and a title), it displays an "Add" button which, when clicked, will add the resource.

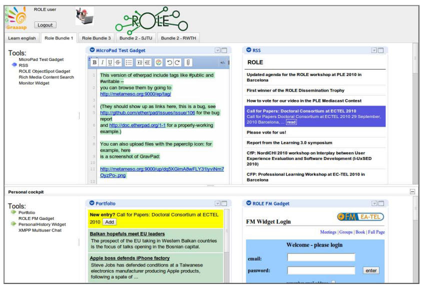
5.3 Displaying on Map

The map widget also receives the event, but handles it differently from the portfolio widget. Instead of requiring the user to click a button, it automatically moves the map to the news story's coordinates, since the map regards this as an action without side-effects.

6 Conclusions

In our test implementations within the Responsive Open Learning Environments (ROLE) project, we are beginning to see how the described user interface, together with the Open Application initiative, allows widgets to be more focused. Instead of a widget providing all functionality by itself, it relies on other widgets providing extra functionality (see figure 2).

The usability concerns (i.e., "just works", automatic, and the user being in control) appear to be well covered. However, further usability issues, that arise in testing, will need to be explored, especially regarding the balance between communication being automatic and the user being in control. After upcoming



This version of etherpad include tags like #public and #writable --
you can browse them by going to http://metameso.org:9000/ep/tag/

(They should show up as links here, this is a bug, see http://github.com/ether/pad/issues/issue/106 for the bug report
and http://doc.etherpad.org/1-1 for a properly-working example.)

You can also upload files with the paperclip icon: for example, here
is a screenshot of GravPad:

http://metameso.org:9000/up/dq5XGimA8wFLY31lyvlNm7OyzPo-.png

ROLE

Updated agenda for the ROLE workshop at PLE 2010 in Barcelona

First winner of the ROLE Dissemination Trophy

How to vote for our video in the PLE Mediacast Contest

Call for Papers: Doctoral Consortium at ECTEL 2010
Call for Papers Doctoral Consortium at ECTEL 2010 29 September, 2010 Barcelona, ... read

Please vote for us!

Report from the Learning 3.0 symposium

CfP: NordiCHI 2010 workshop on Interplay between User Experience Evaluation and Software Development (I-UxSED 2010)

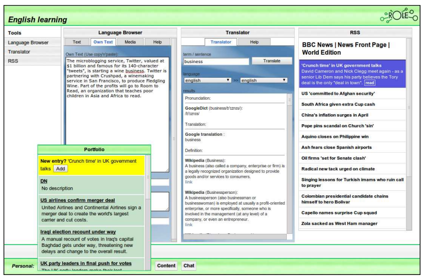
CFP: Professional Learning Workshop at EC-TEL 2010 in Barcelona

New entry? Call for Papers: Doctoral Consortium at ECTEL 2010 Add

Balkan hopefuls meet EU leaders
The prospect of the EU taking in Western Balkan countries is the focus of talks opening in the Bosnian capital.

Apple boss defends iPhone factory
Steve Jobs has defended conditions at a Taiwanese electronics manufacturer producing Apple products, following a spate of ...

(a) Our initial implementation.



The microblogging service, Twitter, valued at $1 billion and famous for its 140-character "tweets", is starting a wine business. Twitter is partnering with Crushpad, a winemaking service in San Francisco, to produce Fledgling Wine. Part of the profits will go to Room to Read, an organization that teaches poor children in Asia and Africa to read.

GoogleDict (business/bˈɪznɪs/):
/bˈɪznɪs/

Google translation :
business

Wikipedia (Business):
A business (also called a company, enterprise or firm) is a legally recognized organization designed to provide goods and/or services to consumers.

Wikipedia (Businessperson):
A businessperson (also businessman or businesswoman) is employed at usually a profit-oriented enterprise, or more specifically, someone who is involved in the management (at any level) of a company, or even an entrepreneur.

BBC News | News Front Page | World Edition

'Crunch time' in UK government talks
David Cameron and Nick Clegg meet again - as a senior Lib Dem says his party believes the Tory deal is the only "deal in town". read

US 'committed to Afghan security'

South Africa given extra Cup cash

China's inflation surges in April

Pope pins scandal on Church 'sin'

Aquino closes on Philippine win

Ash fears close Spanish airports

Oil firms 'set for Senate clash'

Radical new tack urged on climate

Singing lessons for Turkish imams who ruin call to prayer

Colombian presidential candidate chains himself to hero Bolivar

Capello names surprise Cup squad

Zola sacked as West Ham manager

New entry? 'Crunch time' in UK government talks Add

DN
No description

US airlines confirm merger deal
United Airlines and Continental Airlines sign a merger deal to create the world's largest carrier and cut costs.

Iraqi election recount under way
A manual recount of votes in Iraq's capital Baghdad gets under way, threatening new delays and change to the overall result.

UK party leaders in final push for votes

(b) An alternate implementation.

Fig. 2. PLE user interface implementations.

tests with users, the usability concerns, as well as the the inter-widget communication framework and the user interface, will be evaluated and improved upon. The work will also result in a set of terminology, that can be used for environments with inter-widget communication.

Also, we want to cover the other elements in table 1, and integrate them with the described inter-widget communication, especially cross-browser communication, in order to introduce collaboration between users.

References

1. Ivan Zuzak: List of systems that enable inter-window or web worker communication. http://code.google.com/p/pmrpc/wiki/IWCProjects (accessed 2010-06-26)
2. W3C: Resource Descriptive Framework (RDF). http://www.w3.org/RDF/ (accessed 2010-06-26)
3. The Dublin Core Metadata Initiative: http://dublincore.org/documents/dces/ (accessed 2010-06-26)
4. OpenAjax: Hub 2.0 and Mashup Assembly Applications http://www.openajax.org/whitepapers/OpenAjax Hub 2.0 and Mashup Assembly Applications.php (accessed 2010-06-26)
5. OpenSocial: It's Open. It's Social. It's up to you. http://www.opensocial.org/ (accessed 2010-06-26)
6. Web Hypertext Application Technology Working Group: HTML5 Draft Standard: Communication http://www.whatwg.org/specs/web-apps/current-work/multipage/comms.html (accessed 2010-06-26)
7. Garlan, D. and Shaw, M.: An introduction to software architecture. Advances in software engineering and knowledge engineering 1, 1–40 (1993)
8. Zarandioon, S. and Yao, D.D. and Ganapathy, V.: Omos: A framework for secure communication in mashup applications. Computer Security Applications Conference, 355–364 (2008)
9. Isaksson, E. and Palmér, M.: Open Application. http://code.google.com/p/open-app/ (accessed 2010-06-26)