User-tailored Inter-Widget Communication

User-tailored Inter-Widget Communication Extending the Shared Data Interface for the Apache Wookie Engine

Bernhard Hoisl[1], Hendrik Drachsler[2], Christoph Waglechner[1]

[1] Vienna University of Economics and Business, [2] Open University of the Netherlands

Key words: *e-learning, widgets, personal learning environment, mash-ups, orchestration*

Abstract:

This paper presents a technical solution for an Inter-Widget Communication in Mash-up Personal Learning Environments enabling the possibility to model basic workflows. It explains the technical background of the widget concept and why Inter-Widget communication can be beneficial especially for e-learning. Related approaches towards an Inter-Widget communication are reviewed and delimited to the suggested approach. Finally, the detailed procedures of the Inter-Widget communication are presented on the basis of the Wookie widget engine.

1 Introduction

At the present time, Internet users and lifelong learners can create their personal view on the Internet by using tools like iGoogle or Netvibes. iGoogle and Netvibes are *Personal Environments* that allow their users to add and combine different information sources of the Internet in one environment. They can do so by adding so-called widgets, small-scale applications, to their Personal Environment. These widgets encapsulate information from particular web services, like offering a combined RSS feed from several blogs, a search at W*ikipedia*, overview over latest bookmarks form *delicious*, or content sharing opportunities as provided by S*lideShare*. In that way, users can create a personal view of most interesting information on the Internet. Furthermore, users can follow other users and even networks of other users in their Personal Environment. Thereby, the Personal Environments support non-formal learning and inspired research on Technology-Enhanced Learning (TEL) to create Personal Environments to support learning. These extensions of Personal Environments are called in the literature Personal Learning Environments [7] or Mash-Up Personal Learning Environments (MUPPLEs) [8]. MUPPLEs initially support non-formal learning as they require no institutional background, curriculum structure, and are free of use. Their functionalities focus on the needs of the learners rather than on any institutional needs like student management or assessments procedures. Although, they are most appropriate for non-formal learning, educational scenarios are available where MUPPLEs become integrated and aligned with formal learning scenarios in universities [5]. Even more their existence force universities to open up their environments and services to MUPPLEs [3].

A challenging issues until now is the communication between individual widgets in a MUPPLE. So far single widgets have only very limited connectivity possibilities among each other. Several developing efforts are working in parallel to establish an Inter-Widget Communication (IWC) standard, as for example the W3C Widgets 1.0 working draft, the Java portlet specification, or the Google Gadget-to-Gadget communication framework. However, these specifications do not focus on a user-tailored IWC and are lacking possibilities for modelling learner workflows. In the following text, we present such a user-tailored IWC that enables different communication styles between widgets owned by certain users groups like a teacher and his/her students, thus enabling basic service orchestration.

In the remainder of the paper we first explain the widget concept and introduce the Wookie widget engine that has been modified in order to fulfil our objectives (section 2). Afterwards, we introduce the IWC concept, present related IWC research efforts and give a use case for the impact of IWC for e-learning (section 3). Finally, we present our IWC approach in detail by describing the current Wookie solution and how we extended it (section 4). After that, an example is used for illustrating our work (section 5). We conclude the paper with future research and development needs (section 6).

2 Widgets and the Wookie Engine

Widgets are tailored applications that fulfil a small-scale task. Widgets can be included in any website by an HTML embedding code. Most commonly they are used in Personalized Environments where users can create their own combination of widgets for their personal information needs. iGoogle, Netvibes, and Pageflakes are some examples of Personalized Environments. Widgets are visual interfaces to web services on a server in a Service Oriented Architecture (SOA). A widget consists of a client-side programming logic and a visualisation layer to view information given by a web service. The web service on the server only provides an Application Programming Interface (API) to the widget to access data or other programming-logics. The web services contain no commands to visualize any data, most of the time they only return XML data back to the widgets.

Widgets can be developed in any common client-side language such as HTML, JavaScript, Flash, or as Java-Applets. There are several tools for developers to create widgets like Konfabulator for Yahoo widgets, Dashboard for Apple widgets, or Google Gadgets for Google widgets.

At the moment the different types of widgets require specific widget engines. Thus, a Google widget is not deployable in Netvibes and vice versa. This is a disadvantage that the W3C consortium wants to overcome by a collection of specifications in order to achieve a common widget standard and guarantee the interoperability among the different widget engines. Examples for standardisation are the following two W3C candidate recommendation specifications:

‘Widget Packaging and Configuration’ deals with the zip packaging format of the whole widget file and folder structure, the XML based widget configuration file and some mandatory and none mandatory elements. It specifies also the behaviour and means of error handling for the widget user agents.

‘The Widget Interface’ defines an API that enables baseline functionality for widgets, including the ability to: Access some of the metadata declared in a widget's configuration file, persistently store data relating to a widget instance, retrieve the name and value of preferences, which may have been declared in a widget's configuration document.

We based our work on the Wookie widget engine [1] that already take advantage of many of the W3C recommendations and can therefore be seen as a forecast of MUPPLEs and Personal

Environments of the close future.

Wookie is a Java based open-source widget engine and in an incubator phase at the Apache Software Foundation. It was developed by former EU FP6 project TENCompetence. Wookie as a widget engine is responsible for managing and controlling widgets. This means it allows widgets to be embedded in a wide range of web applications (e.g. Wordpress or Moodle) using different connector frameworks. These applications act as a *widget container* for the deployed widgets. Through the Wookie administration interface new widgets can be made available. The widget engine is also able to serve multiple widget containers. This means with only one Wookie instance different widgets can be delivered to many web applications.

A simplified but illustrative example of adding widgets to a PLE can be seen in Figure 1. A widget is instantiated from a Wookie engine and plugged into an existing PLE where learners can use it. The widget acts as the presentation layer which uses the output of two web-services (A and B) for displaying results. Service A fetches data from a database and from another web-service C and delivers its results to the widget. Service B fetches data from an external source, giving back data to the widget where the results of both services A and B are rendered visible for the learner.

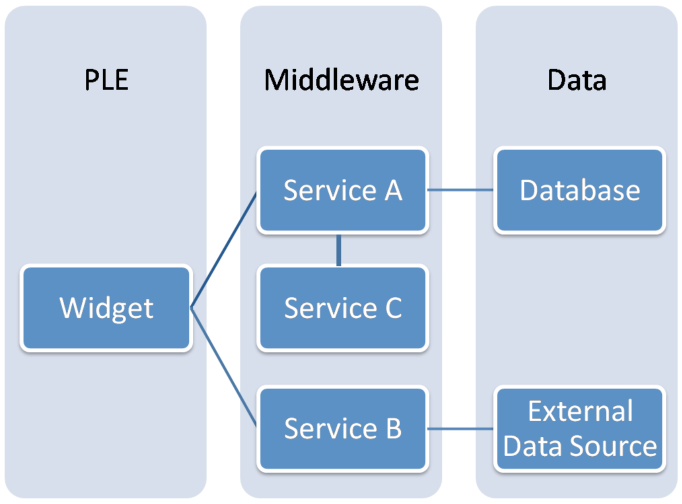


Figure 1: Architecture of adding Widgets to a PLE.

3 Inter-Widget Communication

Widgets can be viewed as separate side-by-side applications integrated and displayed in any widget container (e.g. Moodle, Elgg, or Drupal). As widgets are tailored to one specific task and developed in a modularized way, there is no need to make communication between widgets possible in the first way. This holds as long as a widget container just wants to display different widgets grouped together in one application but with no interaction between them. But if a system designer wants to model, for example, a learning path, or any other workflow within the environment, the widgets need to share their current status and listen to notifications about updates among each other.

3.1 Opportunities for E-Learning

In a learning context IWC extends the highly modularized MUPPLEs with the ability of modelling workflows and orchestration of different information services. It enables user but also system components to send and track data from one widget to another.

In the following section, we will draw a user scenario that illustrates the impact of IWC for MUPPLEs: Bob is a student at the University of Berlin and studies Computer Science in the second semester. After his Bachelor degree he wants to go abroad and finishes his Master at the University of Tallinn. Although, he had several years of English he wants to improve his language skills. Thus, he adds an *English at the workplace* widget to his MUPPLE. He needed to pay for this particular widget as it is provided from a professional content provider. In addition, he adds the *Google English – German Translator* widget to his MUPPLE and a *Vocabulary Trainer* widget. The later ones are offered for free. All of the widgets support Wookie's IWC specification. Therefore, Bob can mark words or even whole sections he does not understand properly and send it to the Google translator and the vocabulary trainer widget. He sends them via the right mouse click context menu to the other widgets. The translator widget immediately looks for a German translation of the marked word and the vocabulary widget shows also related words and offers to save the term in Bobs personal vocabulary list. Bob finds that really helpful as he has not to move between different applications and has all the opportunities he requires to improve his English skills. His Vocabulary Trainer tool also runs on his mobile phone and he can train his personal vocabulary list in the bus to the university.

After he completes his *English at the workplace* course, Bob becomes a member of a Learning Network on English for professionals. The Learning Network applies a tandem learning approach; one day a week the German native speaker teach the English community members and on another day vice versa. The Learning Network members have to correct texts of their peers and also listen to them via a conference tool. Luckily, there is also a widget for the Learning Network available. Thus, Bob can add the Learning Network widget to his MUPPLE and further connect his translator and the vocabulary widget to the new widget over the IWC. As a member of the Learning Network he meets Alice who is also a registered user and they both decide to team up for tandem learning. Alice has a different arrangement of widgets in her PLE and they both share the Learning Network widget only. This is not a problem because IWC makes it possible that they both can communicate over an interface allowing to share data not only between widgets of one user but of many. Therefore, Alice gets notified when Bob has finished correcting her text on *English in the classroom*.

3.2 Related Work

As already briefly mentioned there are different techniques for sharing data between clients and servers in general and for widgets especially. Related work encompass the *myWiWall* portal of the EU FP6 PALETTE project which has taken a first step in making IWC possible and combine it with drag-and-drop facilities [6]. The portal makes use of client-side JavaScript functionalities embedded in the widget host container. However, this approach is lacking the possibility to tailor IWC to a user-defined audience, for example, all widgets of one user or all users that have one particular widget enabled.

Google proposed a Gadget-to-Gadget communication framework [2] where a publisher widget needs to name in the manifest XML file any subscriber widget that is interested in receiving status updates. This approach is very limited and not easily extensible if other widgets should receive updates as well.

Other approaches are dependent on the W3C HTML5 working draft, defining an API for cross-document messaging [4]. Firstly, the draft specification is described more generally and not tailored specifically to IWC and secondly HTML5 is still no web-browser standard and not reliable at the moment.

4 An IWC Solution for Apache Wookie

Wookie does not support IWC by default. Communication is limited to inter-instance data sharing. This means data can only be shared between one specific widget instantiated by users of one widget container, but not, for example, between two or more widgets of one user or between different widgets of different users.

To go beyond the current Wookie approach we developed a branch of the Wookie engine by extending and adding additional IWC possibilities to overcome the current user restrictions. In principle, two functionalities needed to be modified:

1. *The shared data saving strategy:* The location where shared data is saved for a particular widget instance.
2. *The update notification policy:* The way a widget receives a notification to update its data.

Regarding 1, in the default Wookie server, whenever a widget instance submits a shared data update call, the key-value combination gets modified for the certain widget type identified by a GUID (Global Unique Identifier). Normally, this allows for various widgets keeping their individual data although having the same key. But within an environment allowing for sharing inter-widget data it must be ensured that all participating widgets and their instances have access to the same key-value pairs. Therefore, the shared data saving methods had to be changed to have an effect on all widgets that are used by a group of learners. We decided to implement a data cloning method for the Wookie engine to enable the user-tailored IWC.

Regarding 2, three data sharing policies were introduced to send notifications to different groups of users:

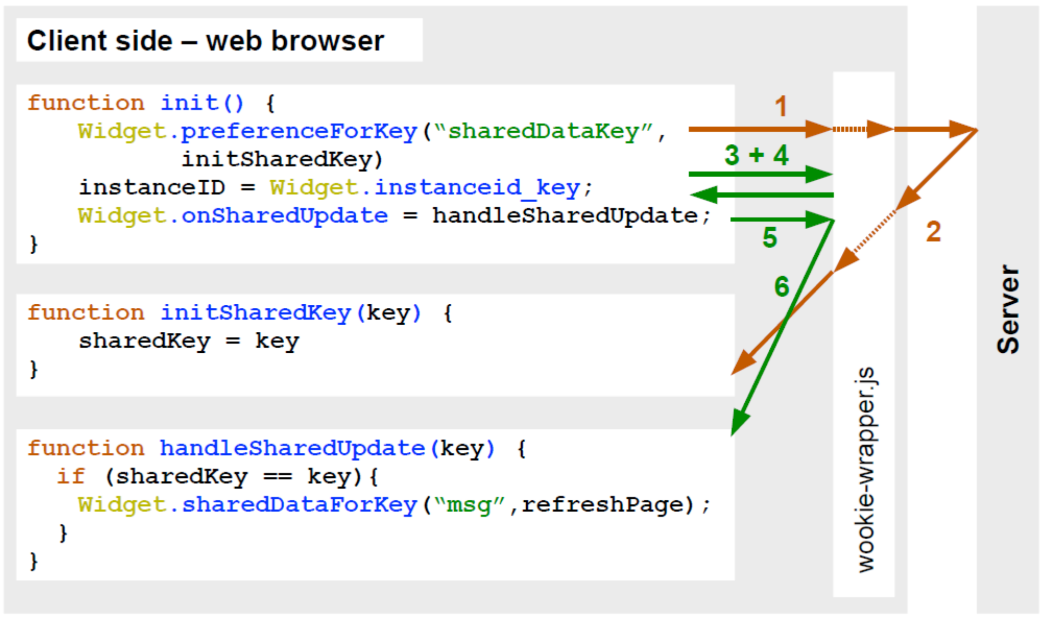
1. “DEFAULT” uses the current and very restricted shared data approach of the Wookie engine.
2. “ALL” allows for an update over all currently instantiated widgets no matter in which container they are located (basically all widgets provided by the Wookie engine).
3. “SAMEUSER” is based on the HTTP-session-ID that contains the user ID and the connection ID to the container application to update all widgets instances a user applies in his MUPPLE.

By adding the DEFAULT value we guaranteed that the initial IWC procedures of Wookie can be used as always. The two new policies allow now for user tailored IWC. The ‘ALL’ policy enables to exchange information between all widgets served by the Wookie engine. Users can adjust information and the whole community receives a notification in their MUPPLE. This can be very supportive for learning communities that need to exchange and discuss information. The ‘SAMEUSER’ policy addresses our use case scenario from section 3.1. It enables exactly this IWC between various widgets in a MUPPLE to achieve beneficial synergies between isolated widgets for personal learning goals. As a result basic workflows can be modelled for individual learners. By having several communicating widgets displayed in their own MUPPLEs, the system designer is able to design pedagogic sound learning paths presented to the user.

Besides implementing these new IWC policies in the Wookie system, the administration web-interface was extended to offer an easy access to the new data sharing options. Therefore, the administrator can change the data sharing policies easily at all times.

5 An Example for using Wookie's IWC Functionalities

In the following section we want to show the communication flow and the applied JavaScript functions we created for the IWC among different widgets. Figure 2 shows with which interface the different functions communicate and how a widget is notified about an update.



```
function init() {
    Widget.preferenceForKey("sharedDataKey",
          initSharedKey)
    instanceID = Widget.instanceid_key;
    Widget.onSharedUpdate = handleSharedUpdate;
}

function initSharedKey(key) {
    sharedKey = key
}

function handleSharedUpdate(key) {
  if (sharedKey == key){
    Widget.sharedDataForKey("msg",refreshPage);
  }
}
```

Figure 2: Code example using the shared data interface.

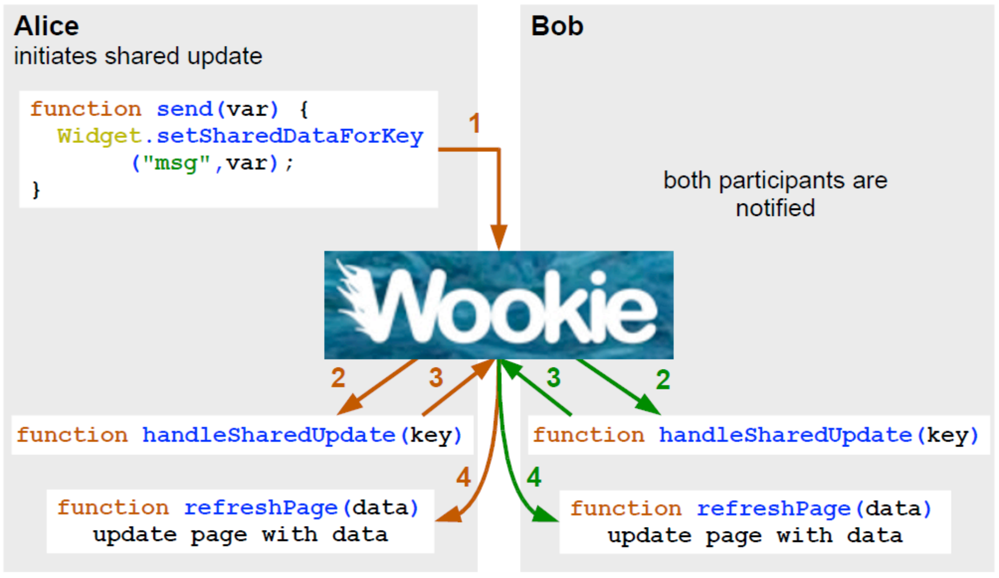
The first action is the request of the *init()* function (1) when the widget is loaded. It retrieves a shared data key for one particular widget instance. This key is necessary to detect update notifications intended for a specific widget instance. Afterwards, the callback function *initSharedKey()* is called (2) with the instance key as parameter. It sets it to the globally available variable *sharedKey*.

In a next step (3), *instanceID* – a unique widget instance identifier – is retrieved and set using *Widget.instanceid_key* function (4). Finally, the update handler is registered and the widget can be updated when new information is available.

Widget.onSharedUpdate() (5) is a function called from the server via remote execution when performing a shared update notification. In this case the function is assigned the value of *handleSharedUpdate()* (6), with the shared data key as parameter. Thus, if on a shared update the update key is identical to the instance's shared data key and the corresponding if-clause in function *handleSharedUpdate()* returns true.

Widget.sharedDataForKey() then retrieves the shared data value for the key msg. At last the callback handler *refreshPage* is invoked and refreshes the widget or updates its content (not displayed here). The widget object (as well as helper functions for browser type and version detection) is provided by Wookie through a JavaScript wrapper file called *wookie-wrapper.js.*

If we assume to have two widgets that need to talk to each other, an exemplary workflow can be seen in Figure 3. There, Alice initiates the shared update by setting a new value to the shared variable *msg* (1). As in this example both Alice and Bob are listening for shared updates (2), also both widgets are retrieving the newly set variable from Wookie (3). Finally, Wookie serves the variable to both clients and a call-back handler (*refreshPage()*) is invoked (4). In both cases, it updates the page with the data retrieved. As mentioned earlier, it is also possible that only one or various amounts of widgets are listening to shared updates and filtering the variables that are supposed to trigger an event, thus making service orchestration and workflow modelling feasible. Events are defined individually for each widget. Therefore, on a shared update, different behaviours of different widgets can be realized.



```
function send(var) {
  Widget.setSharedDataForKey
        ("msg",var);
}
```

```
function handleSharedUpdate(key)
```

```
function refreshPage(data)
  update page with data
```

Figure 3: Sharing data between two different widgets using IWC.

6 Conclusions

Widgets are a novel way of designing MUPPLEs. With the increasing amount of widgets the need for communication among them arise. In this paper, we proposed a new communication method which should overcome current restrictions in IWC approaches. We focused our approach on user-tailored IWC. That means, communication is explicitly controlled through the widget engine and configured to meet users' needs. This is achieved by monitoring data flow of widgets to enable data sharing between widgets of one user, or by widgets that belong to a group of users. We explicitly designed our software to work with the Wookie widget engine although in principle the communication method could be implemented in any other widget server, as well. With our work we try to bridge the gap between a highly individual MUPPLE approach and the possibilities for orchestrating services through IWC. We believe that IWC will contribute to more personalised learning environments, that are tailored to the functionalities users really require for their personal learning goals.

For the close future it is thinkable that there will be widgets available that replace traditional software, like word processor applications (e.g. Microsoft Word or Open Office). Users could use a text writer widget and extend their functionalities with other widgets. With a sustainable IWC solution, the text writer widget could communicate with the other widgets in a MUPPLE and send, for instance, search terms to a translator widget, looking for most suitable pictures or a definition of the term without moving to another environment. This will offer new learning and working experiences and fully address our connected knowledge society future.

Further development work will head in the direction of optimizing the IWC methods between the Wookie server and the client widgets and by ease service orchestration by providing a workflow modelling library. As an example, the current implementation allows only to set the IWC policy for the whole Wookie engine. We are working on developing a per widget based communication policy so that the user can explicitly specify which information should be shared with others. Further improvements will target the notification system, as well, so that users, for example, are notified if an administrator switches between one shared data policy to another.

Acknowledgements

This work has been financially supported by the European Commission under the ICT division of the 7th Framework Programme in the project 'Language Technologies for Lifelong Learning' (LTfLL).

References

1. Gardler, R. (2010): Welcome to Apache Wookie, http://incubator.apache.org/wookie/, last accessed on 2010-06-17.
2. Google (2010): Gadget-to-Gadget Communication (Deprecated), http://code.google.com/apis/gadgets/docs/pubsub.html, last accessed on 2010-06-17.
3. Hermans, H.; Verjans, S. (2009): Developing a sustainable, student centred VLE: the OUNL case. Paper presented at the 23rd ICDE World Conference on Open Learning and Distance Education including the 2009 EADTU Annual Conference. June, 7-10, Maastricht, The Netherlands.
4. Hickson, I.; Hyatt, D. (2009): HTML 5 - A vocabulary and associated APIs for HTML and XHTML. W3C Working Draft 12 February 2009. http://www.w3.org/TR/2009/WD-html5-20090212/comms.html, last accessed on 2010-06-17.
5. Liber, O.; Johnson, M. (2008): Special Issue on Personal Learning Environments. Interactive Learning Environments, 16, 1.
6. Sire, S.; Paquier, M.; Vagner, A.; Bogaerts, J. (2009): A messaging API for inter-widgets communication. Proceedings of the 18th international conference on World Wide Web, Madrid, Spain, April 23, pp. 1115-1116, ACM New York, NY, US.
7. Taraghi, B.; Ebner, M.; Schaffert, S. (2009): Personal Learning Environments for Higher Education: A Mashup Based Widget Concept. Proceedings of the 2nd Workshop on Mash-Up Personal Learning Environments (MUPPLE'09), Nice, France.
8. Wild, F.; Mödritscher, F.; Sigurdarson, S.E. (2008): Designing for Change: Mash-Up Personal Learning Environments. In: eLearning Papers.
9. Wilson, S.; Sharples, P.; Griffiths, D: (2008): Distributing education services to personal and institutional systems using Widgets. In: Wild, F.; Kalz, M.; Palmer, M. (Editors): Mash-Up Personal Learning Environments. Proceedings of the 1st Workshop on Mash-Up Personal Learning Environments (MUPPLE'08), Maastricht, The Netherlands.

Author(s):

Bernhard Hoisl
Vienna University of Economics and Business (WU Vienna)
Institute for Information Systems and New Media
Augasse 2-6, 1090 Vienna, Austria
bernhard.hoisl@wu.ac.at

Hendrik Drachsler
Open Universiteit Nederland
Centre for Learning Science and Technologies

Valkenburgerweg 177, 6419 AT Herleen, The Netherlands
hendrik.drachsler@ou.nl

Christoph Waglechner
Vienna University of Economics and Business (WU Vienna)
Institute for Information Systems and New Media
Augasse 2-6, 1090 Vienna, Austria
h0204938@wu.ac.at