

Towards Collaborative Portable Web Spaces

Stéphane Sire, Evgeny Bogdanov, Matthias Palmér and Denis Gillet,

¹ École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland,
{stephane.sire, evgeny.bogdanov, denis.gillet}@epfl.ch

² Royal Institute of Technology (KTH), Stockholm, Sweden
matthias@nada.kth.se

Abstract. As two recent trends in Web development, Widgets and mashups, are converging, we claim that there is a growing need to define a Widget Space configuration language to allow building flexible personal learning environments that could be independent of a runtime environment. By integrating the do-it-yourself dimension of a Web portal or Web mashup and the social dimension of the Web, we describe the scenarios that could emerge and we describe the new Web components that would be required to support it.

Keywords: widget, mashup, collaboration, web, design scenario, PLE.

1 Introduction

End users of Widgets portals can compose their own personalized environment. Similarly, end users of mashups can benefit from components or applications created with one of the numerous mashup development environments such as Yahoo Pipes. Hence, it is not surprising that Widgets portals and mashups are amongst the most active areas of development today on the Web. For instance, as of July 2009, the mashups directory ProgrammableWeb (www.programmableweb.com) lists as much as 4000 mashups with 3 new ones registered everyday.

Similarly, the exponential development of social network Web sites shows that the social dimension, to bring people together, is one of the driving force that brings people to the Web.

These two trends are particularly compatible with two crucial outcomes of modern learning systems which are to let people construct their own learning environment, and to share their learning experiences together. For that reason, we claim that the future of Widget portals, mashup platforms, and social networks, is to develop a closer integration so that their users can create Widgets Spaces configurations that they can share at different coupling levels. This article explores the consequences of this idea.

2 Web Space Concept

2.1 Definition

Widget portals nearly always focus on providing a customizable personalized environment where various widgets are selected and organized by the user according to some principle. They provide both graphical layout and preferences for individual widgets.

A common approach, supported in iGoogle, Netvibes, Pageflakes etc., is to take widgets that are often accessed at the same time and put them into a separate tab. This could correspond to a course, a projects, an interest, partaking in organizations etc.

In a similar way, mashup development environments allow to pick up software components which are connected together with different predefined settings. Some components are faceless, they are used to fetch, aggregate or filter data, while some others display input fields to allow user to enter data, and the output of the mashup can be visualized in a composite graphical view. This could support too a course, a project, an interest, etc.

In both contexts, we see the "tab", or the "mashup" as a more conceptual Web Space. This space is made of a several components which are combined together, either graphically and/or through some kinds of event/data-flow wiring. If currently the usage of most Web Spaces is to gather information, from a learning perspective we see a great potential in combining it with a social group to support collaborative activities as we will explain in the next sections.

In this article we see a Web space configuration as a three level information structure. The first level describes how to compose the components of a space (layout, graphical theme, data-flow, etc.). The second level describes the initial and default values of any preferences or properties of its components. Finally, the third level describes all the current values that were changed in the second level by the users since they started to interact with the Web space, and eventually some other user's data which are stored within the space and not in external services.

2.2 Applications of the Web Space Concept

Once we recognize the Web space configuration as a basic template for instantiating a Widget composition or a mashup, and as a unit for storing user's data, it becomes natural to consider some mechanisms to keep it independent from the runtime environment through an adequate configuration mechanism. This supports three scenarios: *portability*, *broadcasting* and *co-editing*.

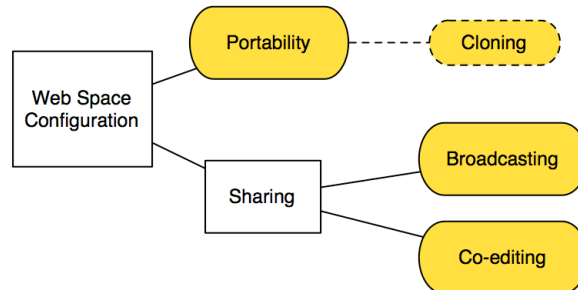


Fig 1. Scenarios enabled by decoupling the Web space configuration from the runtime platform.

The portability scenario allows a user of a Web space configuration, such as a iGoogle page, to move her Web space configuration to another platform, such as Netvibes, for example. We can also imagine that some adaptation to the Web space configuration applies to adapt it to runtime platforms with very different characteristics, such as converting it from a desktop to a mobile platform. The portability scenario also addresses the case when a user invites another user to duplicate one's own Web space, starting or not from a snapshot of one's personal data. This later usage could be called cloning.

The broadcasting and the co-editing scenarios are both scenarios where a connection is established over a long duration by the owner of a Web space that invites other users to share her Web space. In the broadcasting scenario, the owner invites other users to join one's own Web space and to keep them updated when she modifies some personal user's data in this Web space. In the co-editing scenario, the owner invites other users and all the changes to Web space done by any one are merged together so that there is only one Web space configuration.

The portability scenario can be combined with the broadcasting and co-editing scenarios in that different users sharing a same Web space can be accessing it from different runtime platforms. The co-editing scenario can be applied to a single user in the case where this user accesses her Web space from multiple platforms. Moreover, since widgets composition and mashups consist of several components and every component can have its own set of properties and user's data, sharing can happen at different granularity levels. For instance only some components may be shared, only some properties of some components, or some subsets of user's data. Sharing can also happens with different coupling levels depending on the frequency of synchronizations between the different instantiation of the Web space as we will develop in the next sections.

2.3 Current State-of-the-Art

To know to what extent our three scenarios based on Web spaces can be realized today, we have had a look at two widget platforms, namely iGoogle (www.google.com/ig) and Netvibes (www.netvibes.com), and at two mashups

development environments, namely Yahoo Pipes (pipes.yahoo.com) and Afrous mashup engine (www.afrous.com). We first realized that, to some extent, the Widget platforms already supports these scenarios at the individual widget level as explained in the table below.

Table 1. Current support of the proposed scenarios at the Widget level.

	Portability	Broadcasting	Co-editing
iGoogle	Limited	Limited	Limited
Netvibes	Limited	No	Limited

The portability scenario is limited on iGoogle as it works only between users accessing the widget from iGoogle. In that case it appears as a "Send my settings for this gadget" checkbox when sharing it with someone. Netvibes seems also to provide an internal form of widget portability since there is a feature to archive a widget and to restore it at any time from/within Netvibes. Netvibes, as well as iGoogle, also allows you to embed certain widgets by generating a code snippet which can be cut-and-pasted into another environment (see the code below), however in that case the preferences and the configuration are hard coded into the code snippet. This type of portability is also limited to users familiar with HTML. The broadcasting scenario and the co-editing scenarios are supported for certain widgets in iGoogle, they appear as two "View my content" and "View and edit my content" checkboxes when sharing the widget. However this is limited to sharing on iGoogle. A very limited form of co-editing is also available on Netvibes, as a user can access her widget from the desktop and the mobile version of Netvibes.

Widget level Portability scenario: code snippet to embed a Netvibes UWA widget.

```
<script src="../../../UWA/load.js.php?env=BlogWidget2"></script>
<script>
var BW = new UWA.BlogWidget(
{moduleUrl:'... /multipleFeeds.php?provider=wiredblogs'});
BW.setPreferencesValues(
{'nbTitles':'7','showTweet':true, 'openOutside':false});
BW.setConfiguration(
{'title':'Wired Blogs', 'height':636});
</script>
```

Table 2. Current support of the proposed scenarios at the Web space configuration level.

	Portability	Broadcasting	Co-editing
iGoogle	Yes, proprietary	No	No
Netvibes	No	Yes	No
Yahoo Pipes	Yes, proprietary	Yes	No
Afrous	Yes, proprietary	Yes	No

The portability scenario is supported on iGoogle and Afrous as they both allow to save and to import their Web space configuration into a file in different proprietary formats (respectively a gadgetTabML or a JSON file). Yahoo Pipes gives the possibility to clone a pipe from a mashup gallery to edit it.

The broadcasting scenario is supported on Netvibes as users can create one public profile page that their friends can see and which is synchronized with their changes. It is also supported on Yahoo Pipes and on Afrous. On Yahoo Pipes mashups can be embedded into "badges" which can be installed on other platforms through a code snippet that starts a Javascript runtime library (see code below). Similarly an Afrous mashup can be embedded through a Javascript runtime library, as the full mashup engine is a Javascript application. For Yahoo Pipes and Afrous, we consider the embedding as broadcasting as only a reference to the mashup is embedded, hence any modification made by the owner will be reflected in the embedded pipe. This applies to the data-flow composition and constant values parts of the configuration, and not to the user's data, as the concept of user's data (or preferences) is not explicit in these mashup platforms.

Code snippet to embed a Yahoo Pipe data mashup as a badge in a host page.

```
<script src="http://pipes.yahoo.com/js/listbadge.js">
  {"pipe_id":"AHkR4ai42xGlKUilZFUMqA",
   "_btype":"list",
   "pipe_params":{"results":"","search":""}}
</script>
```

The current state-of-the-art shows that we have not yet fulfilled all the potentials of Web spaces, although our selection of 4 platforms shows that there is obviously a move into that direction. To go further, we will suggest in the next sections how we could proceed to define a common configuration language and an underlying architecture that would allow to fully support our 3 scenarios.

Currently the most advance move into that direction that we are aware of is the gadgetTabML format that Google is using to export and import personal pages on iGoogle, and that looks like an undocumented internal feature. The OPML file format is more widespread but it only handles feed widgets in tabs, all other widgets and their preferences are lost. The mainstream widget standardization committees, such as W3C, Open Ajax Alliance, and Open Mobile Terminal Platform (OMTP) do not seem to have plan in that direction, and neither do mashup development environment providers.

3 Web Space Configuration Language

3.1 Configuration Language Elements

A Web space configuration language should at least define the following elements:

- *List of widget or mashup components with default or user settings*: the list refers to components which are published openly and which can be easily referenced via URIs. Depending on the component format, the settings may be expressed slightly differently. Unless the component settings are exposed as properties, it is not stored in the Web space. Components may therefore have separate dependencies to cloud services or real world information for user data storage, but in that case they should rely on other services to share these data.
- *Layout of graphical components*: since not all Web space container support the same layouts, the layout should be treated as hints.
- *Event and/or data-flow wiring of components*: mashups are created by connecting their components, this is also becoming part of a widgets composition as mechanisms such as inter-widgets communication and drag and drop become available. These connections or communication channels are described here.
- *Participants list*: it reflects the social context in which learning occurs. In our vision, one owner of a Web PLE built on top of Web spaces will invite other learners to share an initial configuration associated with some learning goals. That initial configuration template will be created by a teacher for instance, and corresponds to a scenario (e.g. "get to know each other"). The list of participants defines the scope for broadcasting or co-editing when the space is shared.
- *Sharing list*: in order to broadcast or co-edit a space, the sharing list defines precisely the basic units of sharing such as individual properties of a component, a full component, or other properties such as the position of each components in a given graphical layout for instance. We suggest as a sharing policy that all the properties which are not shared can be changed by each user independently of the others.
- *Refresh rate list*: programming patterns such as Reverse Ajax allow to develop notification mechanisms in such a way that changes to shared properties can be notified synchronously. The alternative is to refresh them only on page reload. The refresh rate list indicates the preferred mode of update for each shared properties.

A configuration can be seen as a container that points to external configuration elements, such as the widget configurations. This is a template that supports the creation of a new Web space from external resources. However, the configuration should also store all the shared settings which have been updated by the users in order to distribute them to the group. In this regards we currently describe these settings as properties (i.e. key/value pairs), which is compatible with the practices in most widget and mashup platforms. It is to notice also that in today's environments, these properties have many different usage. Some properties serve as user interface preferences, some others as widget state variables, and finally some others as application data storage. Thus the interest of broadcasting or co-editing a space may depend greatly on the way its components define and use properties.

Finally it appears that we have three orthogonal dimensions in the configuration: participant lists, sharing rights over a property, and refresh rate. The combination of these three dimensions can be used to achieve different styles of user interface coupling [4].

3.2 Configuration Language Syntax

As an exercise we have started to imagine how we could extend the gadgetTabML configuration files so that they can configure different broadcasting and co-editing scenarios. Our proposition is to use a new XML element and 3 new XML attributes (see table below) which can be set on the different elements of a host configuration language to decide who can see them, how they are shared and at which refresh rate.

Table 3. Proposed additions to an XML configuration language to support our scenarios.

Attribute or Element	Value	Meaning
<Participants>	user identifiers (e.g. email)	a list of persons sharing a Web space configuration
@participants	user identifiers	a list of persons viewing a particular component
@sharing	BROADCASTING (a) EDIT (b) NO (c)	type of sharing, (a) means only the owner can change the value, (b) means everybody can change it, (c) it's a private property
@refresh	SYNC (i) ASYNC (ii)	delay before updating the shared properties, (i) means synchronously, (ii) means on page reload

The following example shows a space configuration with 4 widgets that Alice has shared with Bob, Charlie and Dave which are declared in the <Participants> element. According to that configuration Alice broadcasts the xmlUrl property of the first RSS widget (i.e. she imposes the feed). This is declared with the sharing attribute set to "BROADCAST" on the <ModulePrefs> declaration of the property. The updates that Alice will make to the feed URL will be broadcasted asynchronously (i.e. on page reload) because of the refresh attribute set to "ASYNC".

Alice has chosen to display the GoComics widget only in the view of Bob. This is specified with a participants attribute on the <Module> that declares the GoComics widget. Alice has also chosen to impose the value of the "mycomics" <UserPref> property to Bob (attribute sharing set to "BROADCAST") so that she can show him her own choice of comics.

The second RSS widget is visible by everyone and everyone can define his own settings because there is no sharing attribute defined. Finally, all the properties of the Weather widget user preferences, except the "temperatureUnit" (attribute sharing set to "NO"), are shared and can be edited by everyone because its sharing attribute is set to "EDIT", and updates are synchronous.

Extended GadgetTabML file for usage as a Web space configuration (some namespace declarations and some URLs have been removed for simplification).

```

<GadgetTabML version="1.0" xmlns=".../GadgetTabML/2008">
  <Tab title="Accueil" skinUrl="skins/sampler.xml">
    <Layout iGoogle:spec="THREE_COL_LAYOUT_1" />
    <Participants owner="alice">alice@kth.se
bob@epfl.ch charlie@epfl.ch dave@kth.se</Participants>
    <Section>
      <Module type="RSS">
        <UserPref name="numItems" value="3"/>
        <ModulePrefs xmlUrl="http://..."
          sharing="BROADCAST" refresh="ASYNC"/>
      </Module>
      <Module type="REMOTE"
        participants="alice@kth.se bob@epfl.ch">
        <ModulePrefs url="http://gocomics/.../gc.xml"/>
        <UserPref name="defaultIdx" value="1" />
        <UserPref name="selectedTab"
          value="comments" />
        <UserPref name="mycomics"
          value="9|32" sharing="BROADCAST"/>
        <UserPref name="favoritesIdx" value="0" />
      </Module>
      <Module type="RSS">
        <ModulePrefs xmlUrl="http://..." />
      </Module>
      <Module type="WEATHER" sharing="EDIT"
        refresh="SYNC">
        <UserPref name="temperatureUnit" value="F"
          sharing="NO"/>
        <Location name="New Orleans" country="US"
          language="en" />
        <Location name="San Diego" country="US"
          language="en"/>
      </Module>
    </Section>
    ...
  </Tab>
</GadgetTabML>

```

This is a simple proposition to map the participants, sharing and refresh rate dimensions onto a host XML-based Web space configuration language. Of course there are several directions in which the proposition can be refined. For instance, we could imagine a syntax to declare if users can add new widgets/components, if they are shared, and if they can be removed from the space. Similarly we could define a

syntax to define some other configuration elements that could be shared such as the layout. Finally, other extensions are necessary to allow finer grain access control models with sub-groups of users with broadcasting capabilities, and not just use the owner/others dichotomy.

4 Architecture for Collaborative Portable Web Space Configurations

Usually widget and mashup platforms are divided between a server component, that we call the engine, and a client-side component, that we call the container. The engine is mainly responsible for maintaining the Web spaces configurations, usually in an internal database. The widget or mashup components source code may also be installed on the engine server, or they may be available from third party servers. The container is responsible for visualizing the Web space inside a browser, or maybe within another Web space in case portability is offered outside of the platform with compatibility Javascript libraries as we have described in the state-of-the art.

We propose to add a new element to this architecture, namely, a Configuration Server. This component is required to make a Web space portable, or to start sharing it. To simply clone a Web space, an engine just needs to copy the current state of its configuration to the configuration server. To start sharing it, the engine needs to communicate with this configuration server to maintain the configuration up to date as defined by the participants, sharing and refresh settings of the configuration.

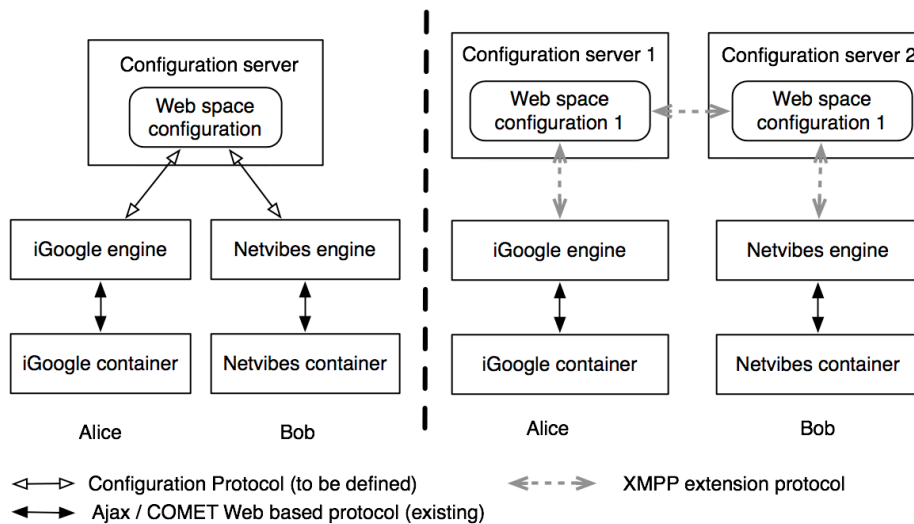


Fig 2. Two visions of the Portable Web Spaces architecture.

This architecture admits two variants. In the first variant, a single, centralized configuration server has been setup to allow two users to share a configuration from

different widget platforms. In the second variant, the configuration data is hosted on different federated configuration servers. The second variant is more flexible in that it allows portability or sharing between platforms that may be connected with different configuration servers, or to have users with different configuration servers in case they need to have an account on it before using it.

5 Implementation Issues

In this section we simply outline some ideas concerning the possible implementation of a Web space configuration server.

The Web space configuration server only needs to manage configuration data, thus it can be implemented with any kind of database. To support sharing, it must also propagate data updates. One solution for synchronous updates is an extension of the XMPP protocol: Google Wave Federation Protocol [1]. In that scenario, the configuration server would implement/utilize this protocol and create a new wave for every Web space configuration. Then, every property of a configuration with a synchronous refresh rate could be saved as a new wavelet into the Web space wave. The participants list of the space would be copied to the participants list of the wave. If the refresh rate is asynchronous the property would simply be saved into the configuration server database.

The engine would communicate with the configuration server with a protocol to be defined, maybe using the Google Wave client-server protocol [2], not only for synchronous properties but also for asynchronous ones. In all the cases, the implementation requires to adapt all the existing platforms so that they can use an external configuration server instead of their internal database.

The container would communicate with the engine using COMET or Reverse Ajax methods or higher level APIs such as the Realtime Gadgets API [3] or HTML 5 Server-Sent Events [5].

Then, two options are possible to manage the sharing policy of the different properties. The lightest solution, that would not require a modification of the engine, is to manage the sharing mode at the configuration level. However, this would not allow to support the broadcasting mode, as everyone could actually change the property and not only the owner. The second option is to manage it at the engine level, so that it prevents users from modifying a shared property if it is broadcasted and they are not the owner. In both cases it would also be necessary to modify the container part of the platform, since the user interface should provide specific affordances and feedbacks to the users so that they know what is shared and how. To some extent the widget and graphical components developers too should also take this into account. Of course, the widget or component API must also be modified to take into account property changes triggered by remote users, for instance through a callback mechanism.

Regarding the social dimension, which is introduced through the participants list, a potentially interesting solution could be to delegate the management of groups of people to an external service such as a kind of OpenId but for the groups, let's call it

GroupId. It should go beyond the definition of a simple list of friends, as defined in an OpenSocial container, to support the management of several groups and subgroups a user belongs too. A group managed by a GroupId server could be declared as a participants list in a configuration through a GroupId URL. This way it should be easier to support many collaboration models, for instance to define open/public participation lists, without changing the Web space configuration language.

6 Wrap up example

The following example shows a concrete application of portable collaborative Web spaces. A teacher has a class with five students. The task of the students is to develop a computer program collaboratively. The program should consist of four main blocks, each solving a particular problem, and a block that combines the code to solve the task. He decides to use the following settings for students: a co-edited widget, that contains the final code, and a widget for every user, where the particular task is specified and a student can do some unit testing on his own code. Thus, the Web space contains these six widgets.

The co-edited widget is available to all users and access to the rest five widgets is limited to one user. Even though the teacher decides to create his Web space in iGoogle, some of the students prefer to use Netvibes. Since the Web space configuration is available to all Web space servers, students can use any environment they want. The changes to the co-edited widget are propagated synchronously to all the users and the teacher. Later teacher discusses the successful class with his fellow teacher from another university. The latter likes the idea, and in order to try it out he receives the clone of the Web space configuration to try it in his Pageflakes portal.

7 Conclusion

In this article we have introduced the notion of a Web space as a basic unit for storing settings and personal data of a Widget composition or a mashup, which can then be independently managed by a configuration server. We have shown that this decoupling between the configuration and the execution of a Widget composition or a mashup enables portability, broadcasting and co-editing scenarios. As we have illustrated, the current state of advancement of the Web is not far from realizing this vision. Thus, we invite every interested party in agreeing on the standards and formats that would allow to construct such a configuration server, maybe on top or as extension of existing technologies such as the Google Wave Federation Architecture or some OpenID extension.

Acknowledgement

This work has been co-funded by the European Union under the Information and Communication Technologies (ICT) theme of the 7th Framework Programme.

References

1. Baxter A., Bekmann J., Berlin D., Lassen S. and Thorogood S.: Google Wave Federation Protocol Over XMPP, July 21 (2009), online: www.waveprotocol.org/draft-protocol-spec
2. Bekmann J., Lancaster M., Lassen S. and Wang D.: Google Wave Data Model and Client-Server Protocol (2009), online: www.waveprotocol.org/whitepapers/internal-client-server-protocol
3. Google: Realtime Gadgets API (2009), online code.google.com/apis/talk/gadgets_realtime.html
4. Dewan, P., Choudhary, R.: Flexible user interface coupling in collaborative systems. In: Proc. of ACM CHI'91 Conf., pp. 41–49 (1991)
5. World Wide Web Consortium, Server-Sent Events, Hickson I. (Ed.), Editor's Draft 7 August (2009), online dev.w3.org/html5/eventsource/