An Infrastructure for Intercommunication Between Widgets in Personal Learning Environments

Tobias Nelkner

[1] Computer Science and Education, University of Paderborn,
Fürstenallee 11, 33102 Paderborn, Germany, tobin@upb.de

Abstract: Widget based mashups seem to be a proper approach to realise self-organisable Personal Learning Environments. In comparison to integrated and monolithic pieces of software developed for supporting certain workflows, widgets provide small sets of functionality. The results of one widget can hardly be used in other widgets for further processing. In order to overcome this gap and to provide an environment allowing easily developing PLEs with complex functionality, the based on the TenCompetence Widget Server [1], we developed a server that allows widgets to exchange data. This key functionality allows developers to create synergetic effects with other widgets without increasing the effort of developing widgets nor having to deal with web services or similar techniques. Looking for available data and events of other widgets, developing the own widget and uploading it to the server is an easy way publishing new widgets. With this approach, the knowledge worker is enabled to create a PLE with more sophisticated functionality by choosing the combination of widgets needed for the current task. This paper describes the Widget Server developed within the EU funded IP project Mature, which possibilities it provides and which consequences follow for widget developer.

Keywords: PLE, SOA, mashup, informal learning, personalized learning, widgets.

1 Introduction

The term *Widget* is the short version of „window gadget“ and has been applied during Project Athena [2]. A widget is usually a window the user can interact with and provides a small set of functionality. Widgets are nowadays well known and used in very different contexts like in a WeBlog to aggregate a RSS feed or for weather information on the operation system like in Windows Vista or Mac OS X. The last years widgets became very important in the research area of computer supported education and (informal) learning [3]. Because of their small set of functionality the learner can personalise its learning environment and choose exactly the one he or she needs to fulfill the current task. Therefore, many environments have been developed to mashup widgets in order to allow learners creating their Personal Learning

Environment (PLE), for example ELGG[1]. A PLE is not a formal educational concept but one for computer supported work integrated learning [4]. Hence, within working processes also more complex tasks have to be fulfilled by the user and therefore need to be supported by the PLE. As Attwell mentioned, a PLE has to support learning among others by aggregating and scaffolding, sharing, and reflecting knowledge with a key focus on communication and collaboration [5]. Bringing the mashup concept of widgets and the requirements on a work integrated PLE together, intercommunication between widgets is a possible solution for creating PLEs more effecient, interactive and immersive. Exchanging data enables two or more widgets to create more sophisticated functionality by linking the output of one widget to the input of another one.

The widget server has been implemented within the EU funded IP project MATURE and is presented in this paper. It provides communication channels between widgets and allows also the collaboration of users via their widgets. This server allows running widgets almost independent from platform and programming language as long as a broker to the JAVA programming language is available.

The next section provides a look to related work in order to show up similarities but also improvements compared to other projects and work. Section 3 then explains the concept, different components of the server and its special functionalities. Section 4 closes the report with a conclusion and an outlook.

2 Related Work

Well known environments exist allowing users including, running, and configuring a widget in its mashups, like iGoogle or Netvibes. Both of them provide an own widget definition for inlcuding self developed widgets and they provide a huge repository of existing ones. But both environments lack of the possibility of intercommunication between widgets and data persistence and therefore do not allow to combine the functionalities of widgets to more complex ones. This leads to isolated functionality of small widgets without supporting users in more complex tasks in a workflow. Moreover, the missing possibility of saving data and sharing it in the user‘s community is a barrier for creating new knowledge. Nevertheless, some widgets provide possibility of saving data but neither it is transparent where and how it is saved (private/public) nor is it accessible for any kind of analysis for tasks like expert finding, provision of information on demand or reuse of certain information in another context.
A closed systems like WordPress that is set up quickly on an own server, provides transparency and sustainability. It is easy to add widgets on the user interface but they

[1] http://elgg.org/

also lack of intercommunication. Yahoo has developed the Yahoo Pipes system that allows mashing up and filtering information streams of RSS feeds. However, intercommunication between widgets means additionally exchanging data and firing events on other widgets for instance to update the user interface or run some computations.
Within the EzWeb/Fast EU funded project [6] a SOA based environment has been created that allows the composition of mashups of widgets by providing them in a special repository. The EzWeb composition environment is a user interface that allows graphically connecting widgets in order to create intercommunication by linking the output of one widget to the input of another one. In the backend, this connection results in an orchestration of webservices. The architecture presented here, bases on the idea of providing widget communication facilities and it does not matter in which environment they run, nor in which programming language they are developed. A minimum of overhead in administration shall be achieved. Moreover, widgets shall not only be visualisations of data but also may realise business logic, depending on what the developer wants to provide to the user. It is not the aim to link the underlying data or logical services they shall remain independent.
Another approach is a result of the TenCompetence partly EU-funded project with the Wookie server as one of the outcomes. The Wookie server allows hosting and instantiation of widgets and provides a proxy server that allows access to different kind of services and therefore also to backend services. However, the Wookie server misses also intercommunication of widgets although it provides a data pool where each widget has access to. This allows the group wide use of data (for example for a chat widget) but no communication in the nearer sense, for example point to point. Nevertheless, this architecture has served for a first version of the server presented in this paper. It has been redesigned and completely new implemented in order to concentrate on the special issue of intercommunication between widgets and between one widget and groups of them within the server. Therefore an independence from the user interface has been achieved. The user can choose in which platform his or her widgets shall run, there is no difference running them in iGoogle, WordPress or an integrated environment as long as they can use the provided API for the server.

An infrastructure of a Widget Server that hosts widgets, provides communication facilities and transparent access to backend systems for persistence and sustainability is a valuable challenge for improving PLE implementations. In the next section the server concept and its different aspects are described.

3 Widget Server Architecture

In order to describe the server structure, the requirements shall be described before explaining the server concept.

3.1 Requirements on the Server

As described before, the most important aspects are the provision of a widget repository, the possibility of intercommunication between widgets and a transparent access to a backend-system for storing, retrieving and analysing data.
The widget repository serves the user as a kind of catalogue to find widgets he or she likes to mashup. Therefore, it provides an upload and download functionality where especially the upload is enriched with validation tests on the widget. Download means, the user can start the widget locally in his client, for instance in case of HTML Widgets, in the web browser. Consequently, a user management is necessary in order to provide user bound persistence services as profile management or configuration settings.
The access to a backend system is necessary to provide persistence services to the widgets for storing or retrieving data. Such a backend system can be a middleware like the Knowledge Bus concept [7] but also a set of fix services provided by a certain system.
A communication system allows widgets to get data and events from other widgets, this can be on a direct channel but also via broadcast to all widgets that are interested in a special data set of another widget.

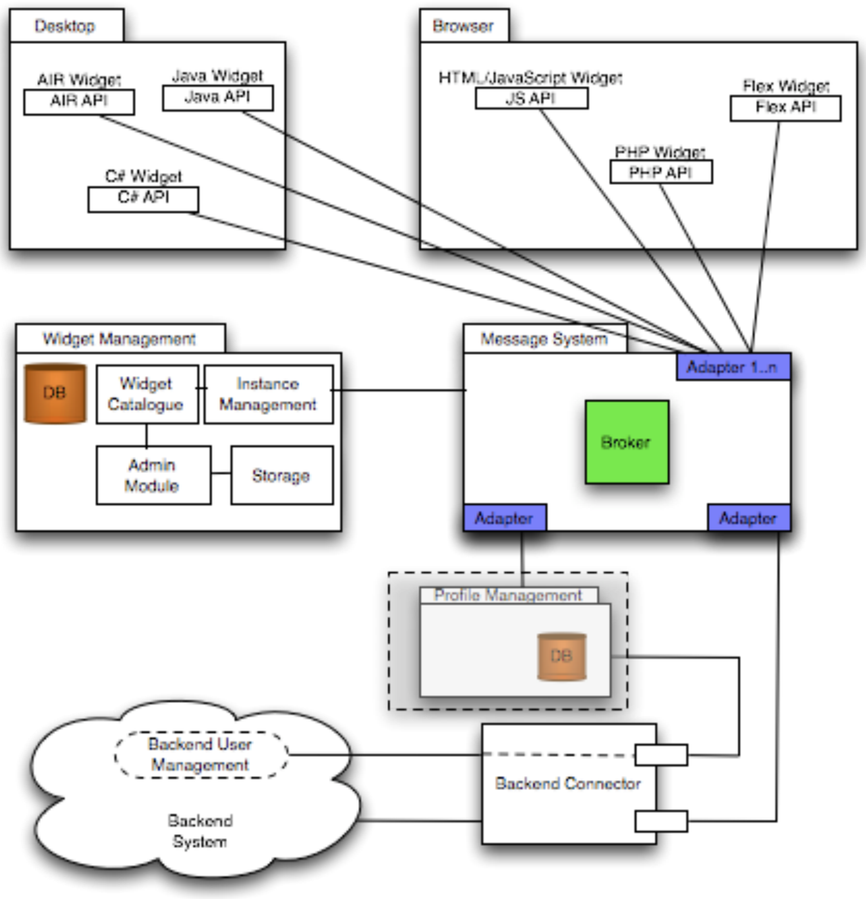

Figure 1: The Widget Server Architecture

3.2 The Architecture

Ongoing from these requirements a first version of the server has been developed based up on the Wookie server of the TenCompetence project. This server already allows the hosting of widgets and undirected information flow. For example, a chat widget is available that allows chatting with all users that opened the widget.

Based on this server, a version was developed that provided a user centric communication channel for all loaded widgets but had also some limitations especially concerning a transparent backend interface and the possiblity of loading widgets developed in different programming languages. For instance, it was not possible to use Flex widgets or JavaFX widgets as the access to the communication channel was not available.

With the experiences of this development, a complete new server has been build that meets our expectations and requirements (see **Figure 1**).

The server is divided into the four parts of Widget Management, Message System, Profile Management, and Knowledge Bus Connector that will be explained in detail in the following.

3.2.1 Widget Management

The Widget Management package of the server contains all modules dealing with the issues of the widget repository mentioned above. This includes hosting, adding and managing running and inactive widgets. Especially the *Widget Catalogue* needs a database to save metadata about hosted widgets and as a backend system is not necessarily available, this database is not outsourced but under the control of this package.

The *Widget Catalogue* holds a collection of the currently available widget types or templates, for example a Weather Widget, Chat Widget, etc. These are then instantiated and executed in the user's client. In addition to this, the catalogue also keeps track of each widget's preferred run-time environment. It is thought to be queried by users' workspaces and provides a list of available and fitting widgets for that workspace. This is important especially to include mobile devices with their special requirements on performance, display size etc.

The *Instance Management* is the most important component of the widget management package of the server. It keeps track of all running instances of widgets in all connected workspaces and automatically purges widgets that have become inactive. It is always the most up-to-date source on the current global state of the widget server, and messaging and administration functions depend on it. As well as keeping track of running widgets, it also manages the state of their communication, for example which widgets are connected to communication units. The messaging system uses this information to distribute messages correctly.

The *Admin Module* performs management functions for the widget management component, usually working on the catalogue. It provides options for catalogue maintenance, widget upload, widget deletion and other administrative functionality. It is intended to be accessed via internal messaging or an authentication-enabled web frontend.

The *Storage Modul* stores widget files, such as Flash, Flex, HTML and JavaScript files. This is basically a shared and protected directory or drive from which the hosted widgets are being loaded via HTTP. This is transparent to the user.

3.2.2 Message System

The Message System is used for widget-to-widget and widget-to-server communication. The *Message Broker* supports point to point communication as well as group based communication models. Groups are created automatically, either because of the input and output paramters of loaded widgets or according to the user's configuration. They are provided by the *Instance Management* module of the *Widget Management* component. All external entities like widgets, an administration frontend or the backend system accessing the server via adapters. The used adapter concept is flexible and allows the connection of all kinds of systems as long as an adapter can be developed that works as a translator between the certain programming languages and JAVA.

3.2.3 Profile Management

The *Profile Management* module is conceptualised for managing server-bound user data that is strictly necessary for the widgets to work. This includes data that is only used to provide essential functions of the widget system and does not necessarily need to be stored in the backend. It includes for example user authentication, default widget positions, saved environment configurations (if available), possibly third-party logins and others. Especially the user management led to the possibility of intercommunication between widgets of different users. This allows to develop communication and collaboration facilities.
Because of its special relevance this module is part of the server as the backend system is not necessarily available and not important for the server and the hosted widgets to run. As long as a backend system is integrated in the server, it makes sense to use this module as a stub and provide the used database as a backend service. Because of this conceptual duality this package is greyed in the image.

3.2.4 Backend Connector

The *Backend Connector* is responsible for providing the access to appropriate services where it is unimportant if this is a middleware encapsulating several services or only one service. All requests and responses are transferred through this connector and that translates the requests to the backend in the specific calls. The connector has to be adapted according to the respective backend system.
Widgets not necessarily depend on the existence of a backend as long as storing or retrieving data from a certain database system is not relevant.

3.2.5 Graphical User Interface

The GUI is the users' client on which the widgets will run. This GUI is connected to the server via an adapter providing access to the communication system. As the adapter is also the broker between different programming languages, the widget developer can choose the one he is comfortable with, which simplifies widget development enormously and decreases barriers working with the server. Moreover, it does not matter if the widgets run standalone in a browser or if they are encapsulated in a closed environment where this is responsible for mapping messages to the certain widgets. Nor is it irrelevant if a widget is running as a desktop client as long it is connected to the server. Consequently it is possible to link browser widgets and desktop widgets interacting with each other. It only depends on the adapter and can be accessed via different techniques, e.g. JMS, DWR and others.

3.3 Relation Between Modules

In order to provide a clearer picture of the interaction between these modules an use case based example of how the server manages the certain requests is shortly presented.
A calendar widget and a weather widget are loaded by including them into iGoogle, interacting with each other and with the backend system. Using widgets in iGoogle is easy as it only contains a link to the *Storage* in the *Widget Management* package. After the browser has loaded both widgets, the server authenticates them at the *Profile Management*. Then a message is send to the *Message System* for registering the widgets in the *Instance Management*, the pool for all running widgets. Furthermore, they are registered to communication group that allows them to talk to each other. This is either realised by connecting widgets automatically according to their input and output format or manually by the user. The latter one is only possible as widget developers have specify the input and output format so that the server can map them. If the user now creates a calendar entry with a certain place, the calendar widget sends a message to the server that distributes it to the common group. The weather widget gets this message, checks if it can interpret the message and updates the display.
By creating the new entry, the calendar widget also sends a save request to the widget server. This is translated into a backend save request to store the calendar dates. The widget server only transfers this message to the backend connector which calls the specific web service or backend system.

4 Conclusions and Outlook

In this paper a widget server has been presented that allows hosting of widgets, intercommunication between widgets and provides an access to a backend system that traces requests to it. It is constructed as a flexible and modular server that allows to connect almost every kind of widgets. Because of the approach of connecting small pieces of functionality the similar based mashup approach of a PLE is fully supported also within the server. This allows for creating more sophisticated and complex functionality by connecting widgets via this server.
With this approach an essential step forward towards personlisable learning environments has been done, especially by providing widget developers a platform for easily creating widgets and using the output data of foreign widgets to provide more sophisticated functionality. Users do not need to care about in which environment, on which client or on which platform they want to run their widgets, the server itself does not have any conceptual limitations concerning this.

But as described above, in order to make the development of widgets as easy as possible but also to be as standard conform as possible, this server concept makes demands on the widgets specification. Hence, the future work concerning this is to provide a specification that allows developers to describe their widgets as exact as possible which is necessary to support linking widgets. Otherwise the mapping of data between widgets would not be possible. Moreover, we have to have a look on the W3C Widget Specification[2] (working draft) and how we can include and validate this standard within our server. In opposition to this W3C standard, we do not limit the server running Web-Widgets. May be a similar specification can be found for desktop widgets and those of other programming languages (e.g. Flex/ActionScript), too.

Acknowledgement

This work was co-funded by the European Commission under the Information and Communication Technologies (ICT) theme of the 7th Framework Programme (FP7) within the Integrating Project MATURE (contract no. 216356).

References

1: S. Wilson, P. Sharples, and D. Griffiths, "Distributing education services to personal and institutional systems using widgets," in Proceedings of the First International Workshop on Mashup Personal Learning Environments (MUPPLE08), September 2008.
2: Arfman, J. M.; Roden, Peter. Project Athena: Supporting distributed computing at MIT, IBM Systems Journal Volume 31, Number 3, 1992.
3: Graham Attwell: Personal Learning Environments for creating, consuming, remixing and sharing, In: David Griffiths and Rob Koper and Oleg Liber (eds.): Service Oriented Approaches and Lifelong Competence Development Infrastructures: Proceedings of the 2nd TENCompetence Open Workshop, Institute of Educational Cybernetics, 2007, pp. 36-41
4: Graham Attwell: The Personal Learning Environments - the future of eLearning? In: eLearning Papers, vol. 2, no. 1, 2007, pp.
5: Graham Attwell, Jenny Bimrose, Alan Brown, Sally-Anne Barnes
Maturing learning: Mash up Personal Learning Environments, In: Fridolin Wild and Marco Kalz and Matthias Palmér (eds.): Proceedings of the First International Workshop on Mashup Personal Learning Environments (MUPPLE08) Maastricht, The Netherlands, September 17, 2008. In conjunction with the 3rd European Conference on Technology Enhanced Learning (EC-TEL'08), Maastricht School of Management, Maastricht, The Netherlands, September 18-19, 2008, CEUR Workshop Proceedings vol. 388, 2008
6: David Lizcano, Javier Soriano, Marcos Reyes, Juan J. Hierro, "EzWeb/FAST: Reporting on a Successful Mashup-Based Solution for Developing and Deploying Composite Applications in the Upcoming „Ubiquitous SOA“, pp. 488-495,

[2] http://www.w3.org/TR/widgets/

7: Hinkelmann, K., Magenheim, J., Reinhardt, W., Nelkner, T., Holzweißig, K., Mlynarski, M.: KnowledgeBus - An Architecture to Support Intelligent and Flexible Knowledge Management. In: Duval, E., Klamma, R., Wolpers, M. (Hrsg.) Creating New Learning Experiences on a Global Scale. Second European Conference on Technology Enhanced Learning, EC-TEL 2007 Crete, Greece, Springer Berlin, ISSN 0302-9743 (Print), S. 487-492.