Development of a Web Based Personal Information Environment

Personal Learning Environment
A Mashup Based Widget Concept

Behnam Taraghi BSc

Development of a Web Based Personal Information Environment

Personal Learning Environment
A Mashup Based Widget Concept

Master's Thesis

at

Graz University of Technology

submitted by

Behnam Taraghi BSc

Institute for Information Systems and Computer Media (IICM),
Graz University of Technology
A-8010 Graz, Austria

03th March 2011

Supervisor: Dipl.-Ing. Dr.techn. Univ.-Doz. Martin Ebner

Entwicklung einer webbasierten persönlichen Informationsumgebung

Persönliche Lernumgebung
Ein mashup-basiertes Widget-Konzept

Masterarbeit

an der

Technischen Universität Graz

vorgelegt von

Behnam Taraghi BSc

Institut für Informationssysteme und Computer Medien (IICM),
Technische Universität Graz
A-8010 Graz

03. März 2011

Diese Arbeit ist in englischer Sprache verfasst.

Begutachter: Dipl.-Ing. Dr.techn. Univ.-Doz. Martin Ebner

Abstract

Due to the enormous growth of distributed applications, services, tools and information resources, especially through Web 2.0 technologies on the World Wide Web (WWW) and also within the universities, it is not easy for end users (learners) to come across existing services, manage and use them in a way that is customized according to their personal needs. Mashups can be a very interesting approach to overcome challenges of distributed (unknown) services. Using mashups in a Personal Learning Environment (PLE) can help to connect resources and applications in an environment customized to the needs of individual users.

On the other hand, personalization is seen as the key approach to handle the plethora of information in today's knowledge-based society. It is expected that personalized information and services will address the needs of the learners more efficiently. The students of tomorrow will regularly have to deal with sharing and merging contents from different sources. Therefore, mashup technology will become a very important means for the realisation of PLE, where the interests and needs of users are the central focus.

In scope of this master thesis, a PLE is developed that is based on the mashup of widgets. The widgets represent the distributed services in PLE and base upon a draft of WWW Consortium (W3C) widget specifications. The PLE is planned to be integrated at the Graz University of Technology as one of the university services in summer semester 2011.

The first sections of this thesis concentrate on the initial approach and the concept of a PLE especially aimed at higher education. The subsequent chapters describe the technological background and structure of PLE, the W3C widget specifications and development of widgets in general, design and usability issues as well as the first prototype in detail. Finally, the conclusion of this written work will sum up the main points of this master thesis, including some ideas for future research and further developments.

Kurzfassung

Aufgrund des enormen Zuwachses an verteilten Applikationen, Dienste und Informationsquellen im Internet und innerhalb der Universitäten, besonders durch Web2.0-Technologien, ist es für die Endbenutzer (Lerner) nicht einfach, die bestehenden Dienste zu finden, zu verwalten und nach ihren eigenen persönlichen Bedürfnissen zu benutzen. Mashups können ein sehr interessanter Ansatz zur Lösung dieser Probleme sein. Durch den Einsatz von Mashups in einer persönlichen Lernumgebung (PLE) kann die Aggregation und Integration von verteilten Informationsquellen und Applikationen realisiert werden. Des Weiteren ist die Anpassung an die Bedürfnisse der einzelnen Benutzer möglich.

Auf der anderen Seite, die Personalisierung ist der Schlüsselansatz für die Bewältigung des Informationsüberflusses in der heutigen Wissensgesellschaft. Es ist zu erwarten, dass die personalisierten Informationen und Dienste effizienter die Bedürfnisse der Lernenden erfüllen. Die Studierenden von Morgen werden regelmäßig Inhalte aus unterschiedlichen Quellen teilen und zusammenführen müssen. Daher sind Mashup-Technologien ein Mittel zur Realisierug einer PLE, wobei die Interessen und Bedürfnisse der Lernenden im Zentrum stehen.

Im Rahmen dieser Masterarbeit ist eine PLE entwickelt worden, die auf einem Mashup von Widgets basiert. Die Widgets repräsentieren die verteilten Dienste in der PLE und basieren auf einem Entwurf der W3C Widget-Spezifikationen. Es ist geplant, die PLE als einen der Dienste der TU Graz im Sommersemester 2011 anzubieten.

Die ersten Kapitel dieser Arbeit befassen sich mit dem Ansatz und Konzept der PLE, besonders an tertiären Sektor. Die weiteren Kapitel beinhalten die technologischen Hintergründe und die Struktur der PLE, die W3C Widget-Spezifikationen und die Entwicklung der Widgets im Allgemeinen, Entwurfs- und Usability-Themen sowie den ersten Prototypen im Detail. Schließlich werden die Hauptthemen dieser Masterarbeit zusammengefasst, wobei auch einige Ideen für die künftigen Forschungs-und Weiterentwicklungen vorgeschlagen werden.

Pledge of Integrity

I hereby certify that the work presented in this thesis is my own, that all work performed by others is appropriately declared and cited, and that no sources other than those listed were used.

Place:	____________________
Date:	____________________
Signature:	____________________

Eidesstattliche Erklärung

Ich versichere ehrenwörtlich, dass ich diese Arbeit selbstständig verfasst habe, dass sämtliche Arbeiten von Anderen entsprechend gekennzeichnet und mit Quellenangaben versehen sind, und dass ich keine anderen als die angegebenen Quellen benutzt habe.

Ort:	____________________
Datum:	____________________
Unterschrift:	____________________

Contents

List of Figures

List of Tables

Acknowledgements

Many thanks go to my colleagues and friends at the division of Social Learning, Computing and Information Services (ZID) and Institute for Information Systems and Computer Media (IICM) who have provided me with help and feedback during the course of my work and my studies.

I especially wish to express my great gratitude to my supervisor Dr. Martin Ebner for his invaluable help, immediate attention to my questions and endless support.

A special acknowledgement goes to Dr. Keith Andrews for providing the Latex template for this written work and allowing the usability tests to be done on the Personal Learning Environment (PLE) first prototype within the scope of one of his lectures.

I would like also to thank Dipl.-Ing. Gerald Till from Institute of Housing for his support and thoughts in the initial design phase of the Graphical User Interface (GUI).

Last but not least, I would like to thank my parents and close relatives for their support during my studies. I dedicate this work to my parents who did not and do not stop believing in me and supporting me with their continuous faith and love.

Behnam Taraghi
Graz, Austria, March 2011

Credits

I would like to thank the following individuals and organisations for permission to use their material:

- The thesis was written using Dr. Keith Andrews' skeleton thesis [Andrews, 2006].
- The core functionality used in Personal Learning Environment (PLE) is adopted from *mywiwall* widget engine implemented in scope of European Palette project [PALETTE, 2008]. Thanks to guys who have been involved in the implementation of *mywiwall* engine. They have done a great job.
- Figure 5.1 is made by Mag.rer.nat. Walther Nagler who has been involved in the creation of first ideas for the PLE concept before the start of this work.
- Figures 5.2, 5.3 and 5.4 are made by Dipl.-Ing. Gerald Till who gave support in the design phase of the PLE for the structure and look & feel of the Graphical User Interface (GUI).
- Some of the widgets that are demonstrated in section 5.4, have been developed under my supervision by some students of Informatics in scope of one of their courses.

Chapter 1

Introduction

" The e-learning application (...) begins to look very much like a blogging tool. It represents one node in a web of content, connected to other nodes and content creation services used by other students. It becomes, not an institutional or corporate application, but a personal learning center, where content is reused and remixed according to the student's own needs and interests. It becomes, indeed, not a single application, but a collection of interoperating applications - an environment rather than a system. "

[Stephen Downes [Downes, 2005]]

Recently, a remarkable transition could be observed in the field of Technology-Enhanced Learning (TEL) as a transition from content-oriented learning or top-down approach (institute-centred; institute as content provider) to a more collaborative-communicative bottom-up approach (user-centred; user as content creator).

It started about seven years ago when Tim O'Reilly pointed to the enormous possibilities of interactions, communications and user-centred approaches on the WWW, Web 2.0, for the first time [O'Reilly, 2005]. Since then, collaboration, content sharing and communication by the use of social software and other networks have been increasing steadily. Nowadays our life is so much influenced by Web 2.0 applications that it might be difficult to perform our daily activities without them. They have been integrated very well as part of the daily life in learning or working environments due to their ubiquitous availability and ease of use [Holzinger et al., 2006] [Klamma et al., 2007]. Twitter[1] as a microblogging platform [Lucky, 2009], and Facebook[2] as a social network are two well-known examples that are mostly used for communication and interaction purposes by the broad public, including learners and teachers. Weblogs [Luca and McLoughlin, 2005], wikis [Augar et al., 2004] and podcasts [Evans, 2008] are further examples that have dominated research in the field of TEL in recent years. There are numerous other works of research that have explored several different possibilities of Web 2.0 technologies, or E-learning 2.0, respectively, as noted by Stephen Downes [Downes, 2005], for teaching and learning purposes in TEL [Ebner, 2007]. In addition to microblogging platforms and social networks that can be applied in various learning and teaching scenarios [Ebner and Maurer, 2008], numerous types of content-sharing services, such as YouTube[3] for videos, Slideshare[4] and Scribd[5] for presentations and documents or Del.icio.us[6] for bookmarks also play an important role in innovative teaching methods and informal learning processes [Mason and Rennie, 2007].

Considering the enormous number of rapidly growing applications intended for the purposes mentioned above, efficient management of these distributed tools can become extremely challenging. Teachers and learners as the main actors in teaching and learning environments may be overwhelmed by the extensive possibilities Web 2.0 tools offer. It may even be difficult to come across the existing services or manage and use them in a way that is customized according to the users' personal needs. Various studies on Web 2.0 technologies at Graz University of Technology (Technical University (TU) Graz) have proven this assertion right and have shown that first-year university students are largely unaware of the existence of numerous Web 2.0 tools [Nagler and Ebner, 2009].

Apart from Web 2.0 applications, the overwhelming effect can also arise with regard to services available within a university or higher educational institutes. Next to some main services that are used by a broad number of students and staff at the TU Graz, there are also many minute services that are unknown to many users, since they are intended for specific kinds of use and not aimed at all user groups. On the other hand, some main tools, such as the administration system (TUGraz online) or Learning Management System (LMS), TU Graz TeachCenter (TUGTC) are growing and being extended in the course of time and provide new functionalities that cannot be handled by all users either.

Mashups can be of great assistance in managing multiple distributed tools, along with handling information and the cognitive overload that comes along with it [Kulathuramaiyer and Maurer, 2007]. Personalization is also seen as the key approach to handle the plethora of information in today's knowledge-based society. It is expected that personalized information and services will address the needs of the learners more efficiently, bearing in mind that the students of tomorrow will regularly have to deal with sharing and merging contents from different sources. Therefore, mashup technology will become a very important means to focus on individual learning needs and to personalize the access to particular information. "The possibility to connect different resources in one environment should help to maintain the overview of all activities. Mashups merge contents, services and applications from multiple websites in an integrated, coherent way" [Tuchinda et al., 2008]. As a result, PLEs offer a new form of personalized learning [Wild et al., 2008]. To overcome the challenge of various distributed resources, the overload of information and the customization of services, the idea of PLE emerged [Schaffert and Kalz, 2009].

In the scope of this master thesis, a PLE based on the mashup of widgets has been developed. The widgets represent the distributed services in the PLE and base upon an extended draft of W3C widget specifications. The widget engine that is applied in the PLE is developed in the scope of the Palette project [PALETTE, 2008]. The PLE is planned to be integrated at the TU Graz as one of the university services in summer semester 2011.

The findings of the initial research, design and development of the PLE at the TU Graz featured in several publications, presented at different international and European workshops as well as conferences that are going to be summarized in this written work.

Chapter 2 discusses the challenges that the PLEs present for higher educational institutions along with the information overflow, whereas other subsequent chapters deal directly with the developed PLE in the scope of this master thesis. Chapter 3 and 4 will describe the technical background and the architecture of the PLE along with the widgets in detail. In chapter 5, the first prototype of the PLE will be presented, which has already been available online since October 2010. Furthermore, this section will give a detailed description of some of the available widgets for the prototype and provide a first expert evaluation of the UI. Finally, chapter 6 and 7 will sum up the main points of this master thesis and present the plans for future research and further developments.

The structure of the Relational Database Management System (RDMS), the class diagram of the client logic and the XML Schema Definition (XSD) of the widgets configuration file are attached in Appendix A for further reading.

Chapter 2

Concept and Challenges

“ Personal Learning Environments are systems that help learners take control of and manage their own learning. This includes providing support for learners to set their own learning goals, manage their learning; managing both content and process, communicate with others in the process of learning and thereby achieve learning goals. A PLE may be composed of one or more sub-systems: As such it may be a desktop application, or composed of one or more web-based services. ”

[Mark van Harmelen [Van Harmelen, 2008]]

The idea of the so-called Personal Learning Environment (PLE) was first introduced by [Olivier and Liber, 2001] as a mashup of different web-based applications. In comparison with traditional e-learning systems, such as the LMSs where mainly formal teaching requirements such as course- or student management are supported, the PLE focuses on users’ individual (formal or informal) needs. Thus the difference actually lies in the role of actors who interact with these systems individually. In a traditional LMS, the teacher manages and arranges the course materials and decides which contents should be provided to the learners. But within a PLE, the learners make their own decisions according to their individual needs and interests. They arrange their learning contents themselves, choose the services they need and manage their learning process individually.

Stephen Downes described the PLE as a future learning environment: ”It becomes, not an institutional or corporate application, but a personal learning center, where content is reused and remixed according to the student’s own needs and interests. It becomes, indeed, not a single application, but a collection of interoperating applications - an environment rather than a system” [Downes, 2005].

Wilson Scott illustrated an image of a Virtual Learning Environment (VLE) [Wilson, 2005] and described the PLE as a user-centred VLE.

Harmelen [Van Harmelen, 2006] argued that the VLEs and LMSs do not handle the individual needs of the learners well. He described PLEs as systems that help learners manage their learning process, starting by defining learning goals up to achieving these goals [Van Harmelen, 2008].

Ron Lubensky [Lubensky, 2006] introduced the general nature of PLEs as a facility: ”A Personal Learning Environment is a facility for an individual to access, aggregate, configure and manipulate digital artefacts of their ongoing learning experiences”. He described PLEs as an intersection of VLEs, Web 2.0 and an expanded view of ePortfolios.

Terry Anderson [Anderson, 2006] compared the advantages and disadvantages of PLEs and LMSs and described PLE as an interface: ”The PLE is a unique interface in the owners’ digital environment. It integrates their personal and professional interests (including their formal and informal learning), connecting these via a series of syndicated and distributed feeds”. He listed the advantages of PLEs as follows:

- Identity: The PLE tools integrate the user's informal identity of their life in an informal setting with formal study.
- Ease of use: PLE can be customized and personalized by the users themselves.
- Ownership: PLE is based on user-centred content or user-owned content and tools.
- Copyright and re-use: As the content and the tools belong to the user, the user can decide on the re-use of the content.
- Social presence: Online-culture reigns in PLE.
- Capacity and speed of innovation: New applications and tools evolve rapidly in PLE.

[Schaffert and Kalz, 2009] summed up the different definitions of PLE as a learning environment where learners can integrate and organize the information, resources, contacts, tools and applications on the WWW and apply them in other online environments.

There are many other works of research that have tried to introduce PLE by using common social softwares such as [Attwell, 2007b] [Attwell, 2007a] [Schaffert and Hilzensauer, 2008]. Due to the numerous possibilities and challenges, especially in the TEL, it seems that no uniform definition can be found for PLEs. The concept of PLEs is still not elaborated well enough to be introduced into approaches used within higher education, as Graham Attwell observed: "Yet for all the talk there was no consensus on what a Personal Learning Environment (PLE) might be. The only thing most people seemed to agree on was that it was not a software application. Instead it was more of a new approach to using technologies for learning. Underpinning a number of the discussions was the issue of what role teachers and institutions would play if learners themselves developed and controlled their own online learning environment." [Attwell, 2007b].

Based on the comparison made by [Anderson, 2006], seven crucial challenges were introduced by [Schaffert and Hilzensauer, 2008] that must be taken into account to switch from a LMS to PLE:

- Role of the Learner: A change from pure consumer to a prosumer or content-producer must take place, which requires some user competences. Users must be able to organize their learning resources and search, find and use the resources that they need.
- Personalization: The user must have the competence to customize their learning environment according to their requirements; self-organisation is vital.
- Content: Competences to search, find, and use the required resources are needed.
- Social involvement: Competences to work with collaborative tools within PLEs are required.
- Ownership: In a PLE, the user is the owner of the data and tools, not the PLE provider. The user must be able to handle that.
- Educational and organization culture: The learning culture in a PLE is different. Users must get used to self-organisation as there are no classes with teacher-oriented instructions.
- Technological aspects : In a LMS, the data can be retrieved from data repositories. In a PLE, certain interoperability strategies are required to aggregate the required data in the PLE.

The list above shows that it would be quite challenging for the learners and teachers to switch from LMS to PLE. However, it is not necessary to substitute any existing e-learning systems by PLE in higher educational institutes. PLE should not act as a substitute for other tools and applications, but rather as an additional environment where learners can apply the services provided by these tools in a more efficient and personalized way. Thus a PLE is not a competitive product for existing tools. Its existence renders present tools and services more valuable since they can be customized according to the user's needs and can be used as a result more efficiently from the user's point of view.

Another challenging point is the probable overwhelming effect within a PLE. A common problem for mashups is the rapidly growing amount of data and availability of tools within the environment. Will the users be able to follow the development of tools and the flow of data in a PLE? The size of entities is a critical factor for the overwhelming effect. If thousands of widgets are provided within a PLE, a mechanism must be found to inform the users about eventual widgets that might be of interest to them. In case of widgets representing social or content-sharing networks, for instance, it is likely that the user is not able to keep up with the mass of information flow in a short period of time. To avoid such problems, a recommender tool could be applied, which can indeed be challenging from the technical point of view (see section 6.2.10).

Chapter 3

PLE Architecture

“ A PLE is comprised of all the different tools we use in our everyday life for learning. ”

[Graham Attwell in ”Qualität im e-Learning” [Attwell, 2007b]]

A PLE that can be seen as a personalized individual website is not a new idea. This idea has been applied in the form of contact widgets in some social networks, such as Facebook, to enable users to get in touch with other registered users on the platform. Nowadays PLEs are gaining increasing attention with the growth of Web 2.0, as described in chapter 1, and Rich Internet Application (RIA) technologies. It allows developers to build more dynamic and stable client-side applications with a flexible GUI and programming logic. The programming logic is most often fully integrated in the presentation layer on the client side, which results in a distribution of server load, reduction of server response time and achieving a higher performance. The server-side logic of the applications is not responsible for the presentation layer anymore. Its only task is to provide clients with the data and resources they need for using an API in a Service Oriented Architecture (SOA).

The goal of a PLE cannot be reduced to being only a platform for accumulating distributed learning applications, used at university or on the Internet. Certainly one of the goals is that students are able to adapt their learning environment to their preferences, so they are able to make their own decisions on which applications they want to use and integrate into their environment. By the same token, each application or service that is integrated into a PLE should be flexibly configurable to meet the individual needs of the student. From the technical point of view, a PLE is a client-side environment RIA, comprised of a mashup of different small independent web applications and services selected by the user [Taraghi et al., 2009a]. These distributed applications are configurable and can communicate with other web applications within the PLE environment. What is more, Hoyer [Hoyer, 2008] introduced some existing mashup tools with different emphases, such as Yahoo Pipes and Microsoft Popfly. Aumüller and Thor [Aumüller and Thor, 2008] described three main components of a mashup application: data extraction, data flow and presentation. They categorize different mashup tools according to one or several of these components.

As it is not possible to integrate the entire set of services into one presentation layer, the PLE server serves as a single entry point to provide the client-programming logic with such small applications or services. These small applications are called widgets.

Widgets are small embeddable applications that can be included in an HTML-based web page or executed on the desktop. This client-side code can be a simple JavaScript, Java-applets or anything that can be embedded in a valid HTML or XHTML document. It entails the functionality to build the GUI of the widget dynamically and the logic to retrieve or update data from services provided by the PLE server as well as remote servers [Taraghi et al., 2009b]. Chapter 4 discusses widgets in general as well as the widget specifications of W3C.

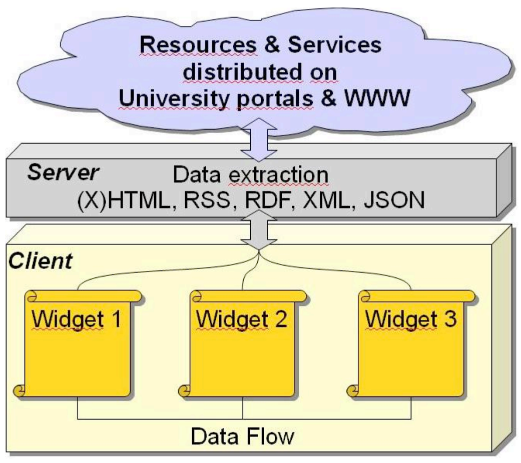


Figure 3.1: Mashup structure of PLE describing data extraction from distributed resources, data flow between widgets and presentation components in PLE.

As mentioned before, personalization is a very important factor for a PLE. Users should be able to customize their learning environment personally and in accordance with their own needs. The customization of GUI is important to give learners a personalized look-and-feel. Besides, the users should be allowed to use and customize the various distributed learning applications and university services on the Web in their PLE. In order to meet these requirements, a mashup of widgets can be used. When short lightweight applications are put into widgets, users can organize and personalize the applications (widgets) that are interesting for them, which they can combine into a vast number of possible mashups [Taraghi et al., 2009a].

The mashups of widgets used in a PLE can be classified as end-user mashups, as described in [Gamble and Gamble, 2008]. The PLE contains a widget engine, implemented in the Palette project [PALETTE, 2008] to load and handle the widgets according to the W3C widget specifications. While data extraction is carried out on the side of the server, the data flow and presentation components are handled by the widget engine on the side of the client, as illustrated in figure 3.1.

Applying widgets in a PLE can have several advantages [Taraghi et al., 2009b]. Widgets represent independent web applications, hence they can be implemented independently from a PLE. The W3C widget specifications, which are explained briefly in section 4.1, introduce a unique standard for widgets. If this standard is applied, it could result in many open-source widgets that can be employed in different PLEs or other learning systems, supporting the W3C widget specifications. Another issue is the distributed knowledge transfer from different servers, along with diffusion. The service used by the widgets must not necessarily be located on the same PLE server. Remote servers provide widgets with corresponding services through their API. Widgets cannot send cross-site requests to remote servers due to security restrictions of the XHR object in browsers. Yet, there are some techniques to bypass this restriction. which are described in detail in section 4.4. In the PLE, a proxy script is used on the PLE server to enable cross-site communication between widgets and remote services. As a result, many different distributed remote services can be provided within the PLE without any technical effort.

Section 3.1 introduces two European projects that have already been implementing W3C widget 1.0 specifications (Packaging and Configuration as well as API). Based on the widget engine implemented in the Palette project, the PLE is designed to be applied at the TU Graz within the framework of this thesis.

In order to know what users can do exactly in the PLE, it is necessary to know about different use cases beforehand. Section 3.2 gives a general description of such use cases.

As mentioned before, the PLE as a whole can be considered as a RIA with a client-server architecture. The separation of tasks between the client and the server makes it possible to work on each tier independently. Sections 3.3 and 3.4 introduce the server and client architecture of PLE in detail. They contain the adapted architecture from *mywiwall* portal and the additional newly implemented extensions as well as the necessary upgrades.

Section 3.5 describes briefly the Widget Development Environment (WDE) that is developed within the scope of this master thesis to make widget development for widget developers on the local host possible.

3.1 Technical Background

Next to the TenCompetence project, the Palette [PALETTE, 2008] is one of the two projects of the European Union (EU) focusing on educational applications. They have already implemented the draft of W3C widgets specifications (Packaging and Configuration as well as API). Refer to section 4.1 for further information.

Wookie[7] is a standalone widget engine which was developed as a part of the TenCompetence project in order to enable coordination of the usage of different external tools in learning activities. It can be integrated into any web application by using the *wookie* widget factory API. Its task is to instantiate the widgets within the platform and render them to the UI of the corresponding web applications. To simplify this integration for existing applications, plug-ins are developed to be used in certain famous web applications, such as Wordpress[8], Moodle[9] and Elgg[10] [Taraghi et al., 2009c]. Unlike Wookie, the widget engine in the Palette project is implemented as a part of the Palette web portal *(mywiwall)* [Taraghi et al., 2009b], which means that the integration of widgets in other applications cannot be performed by plug-ins. In the Palette project, the W3C widget configuration is extended and some additional default user preference values are added. These values may be modified by users to customize the widgets according to their own needs (see section 4.2). Although this extension is advantageous for e-learning systems where widgets can be configured and customized according to their preferences, the *wookie* widget engine cannot handle these widgets, as their manifest file contains extended elements not described in W3C packaging and configuration specifications. But since the Palette preferences are added under a separate namespace, it remains compatible with the W3C specification (see section 4.2.2).

The Palette service portal represents a web portal *(mywiwall)* that users can customize by adding and removing widgets. The W3C built-in widget engine enables the installation and integration of any widget that is compatible with the W3C specification.

Palette has extended the Widgets Interface specifications described in section 4.1.2 as well. Through these extensions, a new way of communication between widgets has been enabled. Widgets can add listeners to events or fire events to trigger certain events in other widgets (see section 4.3.5). This can be graphically realised by a simple drag & drop between widgets. As a very simple example, setting a location in a map widget can fire an event that triggers the other widgets within the web page. A weather forecast widget can adjust its contents after being triggered by the map widget to show the weather status of a selected location on the map.

Next to HyperText Transfer Protocol (HTTP) authentication, Palette also supports a separate widget authentication mechanism in case the widgets are required to be authenticated by third-party services that they make use of or represent (see section 3.3.1).

Palette distinguishes between local widgets, which are deployed within the platform, and remote widgets, which are stored on a remote server. While local widgets are static client-side applications that are compatible with W3C widget 1.0 specifications and implemented in HTML and JavaScript, remote widgets may include server-side programming languages that dynamically produce the widget content. In theory, it is possible to implement any type of widget using both approaches. Looking at a wide scope of learning objects and services together with their eventual dynamic processing requirements

in the background, it becomes clear that the remote widgets provided by Palette can be applied very usefully in this context. The widget variation used in a PLE can be increased and extended to many learning services on remote servers in the form of remote widgets.
Note: The PLE designed in the framework of this master thesis is based on the widget engine implemented in *mywiwall*. It supports the W3C widget packaging and configuration as well as API specifications. Since W3C specifications have often been updated, the widget engine is outdated and needs to be updated to remain compatible with the current version of W3C specifications (see section 4.1).

3.2 Use Cases

Two types of actors are designated for PLE: Administrators and current users. The use cases of each actor are described as follows:

3.2.1 Administrators

Administrators are authorised to organize and manage the provided widgets within PLE. They can install and de-install widgets, update the already installed widgets to new versions (edit widgets), add, edit and remove categories in the PLE portal, and last but not least add, edit and delete users. Moreover, they can view statistics results and the list of users who have been online in a certain period of time.

Statistics data regarding user behaviour in the PLE can be viewed for a certain period of time and include the following cases:

- The number of widgets that have been used by the user at least once.
- *(A)*: The list of widgets that have been used by the user at least once.
- The number of times the user has used the widgets *(A)*.
- The number of times the user has used each widget in *(A)*.
- The average number of times the user has used a widget in *(A)*.
- The number of times the user has been online.
- The period of time the user has been online each time.

Statistics data regarding the usage of widgets in PLE can be viewed for a specific period of time and include the following cases:

- The number of users who have used a widget at least once.
- *(B)*: The list of users who have used a widget at least once.
- The number of times a widget has been used by users *(B)*.
- The number of times a widget has been used by each user in *(B)*.
- The average number of times a widget has been used by one user in *(B)*.

The statistics data regarding user agent in PLE can be viewed for a certain period of time and include the following cases:

- number and percentage of different Operating Systems (OSs) of user agents

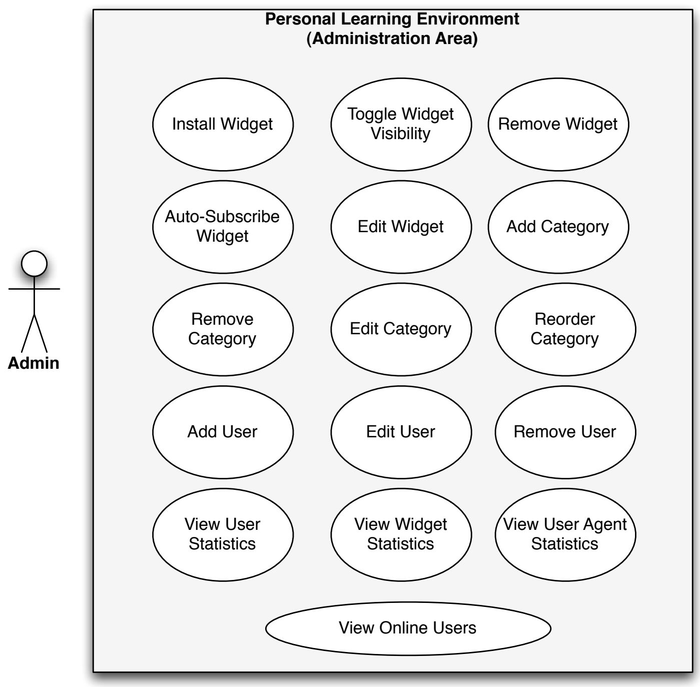


Figure 3.2: Use cases for PLE administrators.

- number and percentage of different browsers of user agents
- number and percentage of different browsers in specific OSs of user agents
- number and percentage of mobile user agents

Figure 3.2 shows the use cases regarding *administrators*.

3.2.2 Users

Users can be either students and lecturers of the TU Graz with a valid TU access account, or external users who have been registered by the administrator to be able to access and use PLE. They can activate (add), remove, search or move the widgets in the UI. What is more, they are also able to add or remove widgets on their personal desktop, which is part of the UI. For more details, refer to section 5.1.

The login to PLE is different for students and lecturers with a valid TU access account. A while ago, the login to TU services and web applications was switched to Single Sign-on (SSO) authentication mechanism. Once the user has acquired authentication in one of the TU services, no further authentication is required to access other TU services. The same mechanism is used to log out. Once the user is logged out of one of the TU services, all other sessions from other services are closed as well. As the PLE is going to be used as one of the TU services, the SSO authentication applies. The SSO at TU Graz is based on the Shibboleth[11] system, where a central Identity Provider (IdP) is responsible for the identification and authentication of users via a HTTP Secure (HTTPS) connection.

Figure 3.3 shows the use cases regarding *users*.

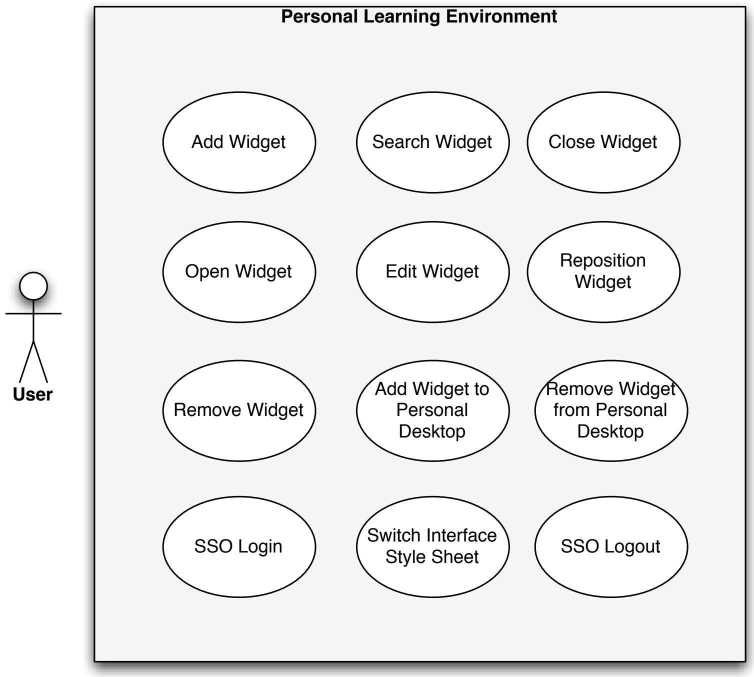

Personal Learning Environment
User
Add Widget
Search Widget
Close Widget
Open Widget
Edit Widget
Reposition Widget
Remove Widget
Add Widget to Personal Desktop
Remove Widget from Personal Desktop
SSO Login
Switch Interface Style Sheet
SSO Logout

Figure 3.3: Use cases for PLE users.

3.3 Server-Side Architecture

As mentioned in the preceding sections, the PLE as a whole can be considered as a RIA with a client-server architecture. The server is merely responsible for the retrieval of data from data resources. The tasks that the server has to fulfil can vary from adding and updating data to local storage (such as user preferences, widgets, etc.) to retrieving data from remote servers. The server scripts act as web services. An Asynchronous JavaScript and XML (AJAX) approach is used to transmit data between the client and the server. The server responds to the client's GET requests with data retrieved from the data storage, either in eXtensible Markup Language (XML) or JavaScript Object Notation (JSON) format.

The server architecture is based on the *ClearFw*[12] framework. *ClearFw* is a framework based on the Model View Controller (MVC) design architecture and is implemented by CRP-HT[13] in Hypertext Preprocessor (PHP) 5 programming language. Figure 3.4 illustrates the client-server architecture that is applied in PLE.

Section 3.3.1 describes the widget authentication possibilities for third-party services. Section 3.3.2 describes the proxy module that deals with the transmission of data between widgets and external resources (services). Section 3.3.3 sheds light on the data storage that is distributed on the file system and on RDMS. Section 3.3.4 describes the server API that is responsible for handling client requests.

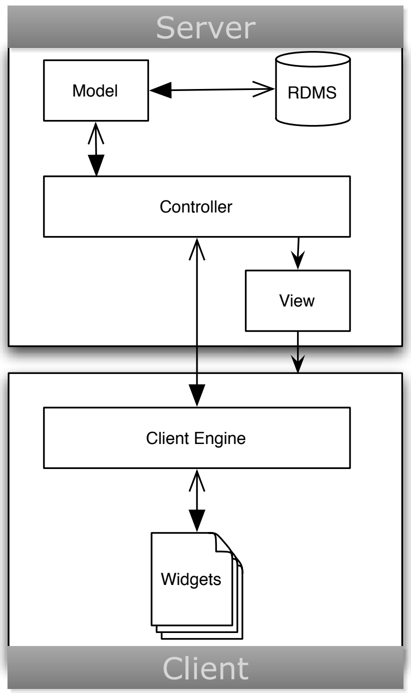


Figure 3.4: PLE client-server architecture.

3.3.1 Widget Authentication

PLE acts as an environment where various services and applications on the WWW and from the TU Graz are integrated in and realized through a mashup of widgets. Contrary to most of the general services on the WWW that are accessible publicly, there are many user-based services which may require authentication, such as SlideShare, Scribd, Facebook, etc. Most of these services provide methods for authentication through their API where the username and password must be submitted as arguments. The corresponding widgets must send the login data (saved as user preference on the PLE server) to the service API through a built-in proxy (see section 3.3.2).

Yet, in many cases no API is provided for authentication. For instance, there can be services that require HTTP authentication. The same holds true for many university services. Most university services are not accessible to the public at all and their use is restricted only to users with a valid TU Graz access account. Since TU Graz users are authenticated in the PLE through SSO, they should be authorised to access to all TU services without the need of repeated authentication for each service. On the other hand, widgets can communicate with remote servers through a proxy. The proxy acts as an independent web client for web services and is not authenticated for TU services; hence it does not own the SSO cookies that the widgets own. Therefore, a widget authentication mechanism is necessary to establish a trust relation between widgets deployed by the PLE server and remote services. In this way, the widget should be authorized to apply the remote service. Furthermore, the user Identifier (ID) in PLE must be transmitted to relevant remote services for eventual retrievals of user-related data.

In order to meet these requirements, the approaches given below are applied.

Secret Shared Service Key (S3K)

The 256 bit *S3K* shared between widgets and remote services is used in a cryptographic algorithm called *Rijndael*. If the widget uses this authentication mechanism, a shared key is generated and saved automatically in the PLE Database Management System (DBMS) at widget installation time [Naudet et al., 2008]. The widget authentication mechanism must be declared as enabled in the widget configuration file to let the installation module know that the widget requires a S3K to communicate with remote services (see section 4.2.2). The generated S3K for the widget can be viewed in the administration area and shared manually with other services. Once a XHR arrives from a widget with an enabled authentication mechanism, the proxy encrypts the username of the user using the widget in the PLE with the S3K of the widget and sets the username along with the encrypted version of it as (user, password) pairs in the HTTP authentication header. The remote service needs to read the HTTP authentication header values and decrypt the password with the same S3K. If the decrypted version is identical with the user, the service provider authorises the user to use the service.

This widget authentication mechanism is applied in the *mywiwall* engine as well.

Randomly Generated Token (RGT)

RGT is another approach that is implemented in the PLE. It rests on a very simple idea. The remote servers must be registered on the PLE server with their main Uniform Resource Locator (URL) and Internet Protocol (IP) address. The client (widget) sends an XHR to the PLE API to retrieve a RGT and the username of the user at first. The tokens are 32 characters long and are generated on the fly randomly. They are valid only for a limited period of time and are unique for each user and each subscribed service. When the widget sends an XHR to the remote service, it sends the retrieved token and username of the user as an additional parameter. The remote service calls the PLE API to verify if the token is valid for the specified username. If the service is subscribed and the token is valid for the specified user, the remote service will respond positively.

HTTP Authentication

PLE also supports normal HTTP authentication, if so required by a web service. In this case, the widget API provides the developers with methods to set the HTTP authentication header with corresponding login data, namely the user and password data (see section 4.3.4), when they send an XHR to the remote web service. The build-in proxy on the PLE server passes the header values on to remote services [Naudet et al., 2008].

3.3.2 Proxy

The server is responsible to retrieve data from external resources upon request of the client (mainly widgets). Due to security restrictions of browsers for XHRs, the client (widgets) cannot send requests directly to remote servers. This feature is most often required by widgets that represent or make use of third-party services on the WWW. To meet this requirement, a proxy module is used.

The proxy acts as a web client for remote web services. It passes on all unmodified HTTP requests and responses from the client to the target services and vice versa. The header values are passed on as well, so that target remote servers can handle the request and the client can parse the response correctly. In case of widget HTTP authentication, the authentication header values (user and password) are passed on as they are in the client request to the target servers. For the shared-key authentication mechanism, provided that it is enabled, the username in the PLE and the encrypted version of it are set as the corresponding HTTP authentication header (user and password) [Naudet et al., 2008].

3.3.3 Data Storage

The PLE requires a DBMS and the file system to fulfil the requirements. These are described as follows:

File system

The file system is needed to save widget archives on the server. Widgets are actually archives of files that are required for each widget to run, such as images, Cascading Style Sheets (CSS), JavaScript and Hypertext Markup Language (HTML) Files. The administrator has to upload the widget to install it on the PLE. The server verifies the widget archive according to W3C widget packaging and configuration specifications, and finally saves it on the file system under ./widget/ directory. The default values for user preferences are read from the configuration file during installation and saved in DBMS for easier processing.
Note: Remote widget archives include only a configuration file that is saved on the file system after successful installation.

Moreover, there is also an ./upload/ directory where miscellaneous files can be uploaded by the administrator. A category icon can for instance be set in the administration area for each category.

Relational Database Management System (RDMS)

For the RDMS, a database abstraction layer is used from Clearbricks[14]. For the current version, only three database drivers are provided to support MySQL[15], PostgreSQL[16] and SQLite[17] RDMSs. To support other DBMSs, corresponding drivers must be provided. The database driver can be set in the config.php file on the server. This is advantageous since it is possible to change the RDMS, if desired, simply by resetting the database configuration in the config.php file.

A detailed view of the database structure is given in appendix A.1. The tables in the PLE database are described as follows:

- *categories*: Contains categories defined by the administrator and their display order in PLE's UI.
- *cops*: Contains user groups that are defined manually.
- *interface*: Contains the activated widget IDs and their positions (column, position) as well as the status (minimized or maximized) in PLE's UI for each user.
- *interface_dashboard*: Contains the activated widget IDs and their positions (column, position) as well as the status (minimized or maximized) on PLE's personal desktop UI for each user.
- *interface_preferences*: Contains user preferences of users for each widget.
- *login_data*: Contains statistic data about the logged-in users such as session, login and logout time, metadata about user agent (browser, OS, browser version, etc.).
- *tags*: Contains tags submitted to the system.
- *tracks*: Is used to collect data in relation to the use of widgets for usability purposes. It contains the widget IDs and the timestamp that are used by each user.
- *users*: Contains the list of registered users in the PLE and their membership of user groups.
- *widgets*: Contains the list of installed widgets in the PLE and their membership of user groups.
- *widget_tags*: Contains tags submitted by each user for each widget.

Figure 3.5 demonstrates the *one to many* relations between the described tables in the database.

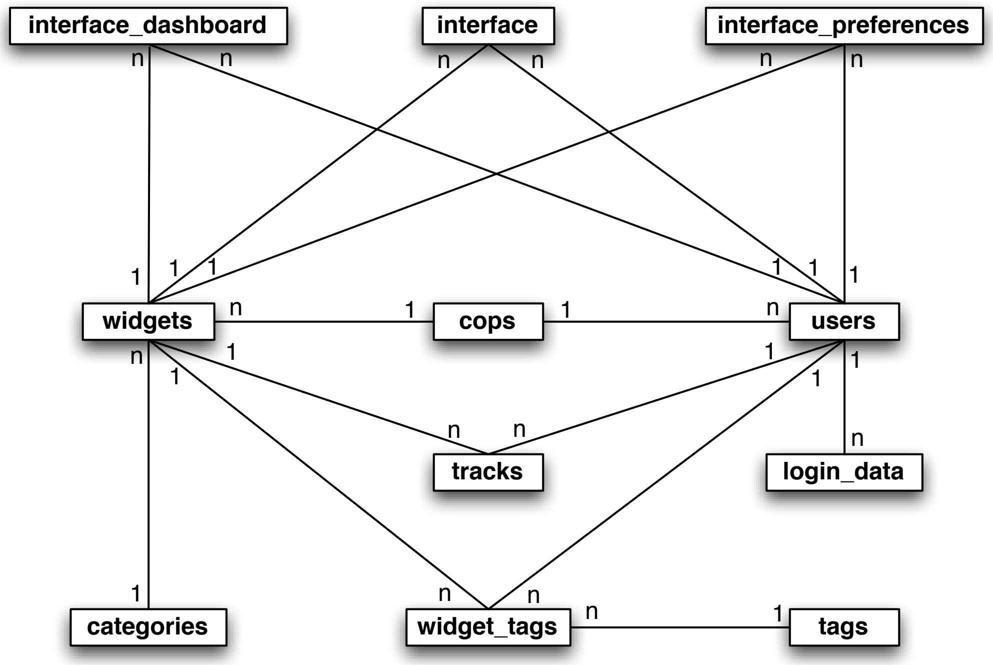


Figure 3.5: *One to many* relations in PLE database.

3.3.4 Application Programming Interface (API)

As mentioned in section 3.3, a PLE is a RIA with a client-server architecture. The server acts as a web service for the client. The client sends GET AJAX requests to the server and the server responds to the client by sending the required data either in XML or JSON format. The only exception is the administration area. Since the administration area is normally used by one or a couple of users as administrators, it does not rely on RIA principles. It is realized as usual classic web development. The content is generated dynamically and the whole generated layout is pushed into the browser as HTML markup. To view the services that are provided in the administration area, refer to the use cases in section 3.2.

The services described below are provided by PLE API for the client's XHRs.

Widget-related services

This API is used for client requests that are related to widget operations (see section 3.4):

- *updateInterface*: Updates the UI status of the user in the database when a widget is added to, deleted from, moved, opened or closed in the user's UI.
- *updatePDInterface*: Is the same as *updateInterface*, but just for the user's personal desktop.
- *getCategories*: Returns the list of categories.
- *getInterface*: Returns the complete UI of the user (list of widgets, widget positions in user's UI, etc.).
- *getManifest*: Returns the XML manifest file of the widget with user preferences attached.
- *getTags*: Returns all tags already submitted to the system.

- *getUserTagsForWidget*: Returns the tags submitted by the user for the widget.
- *getWidgetList*: Returns a list of widgets that are tagged by a keyword or exist within a category, or both.
- *updatePreferences*: Updates user preferences for a widget.
- *isUsingAuthentication*: Returns if the widget applies the HTTP authentication mechanism.
- *getWidgetsByCategory*: Returns widgets within a category.

User related services

This API is used for client requests related to user operations (see section 3.4):

- *shibbolethAutoLogin*: Signs in the user if the user is already logged in at a Shibboleth service provider.
- *authenticate*: Performs user authentication.
- *logout*: Performs user logout for external users.
- *shibbolethLogout*: Performs logout for users with a valid TU account.
- *updateStyleSheet*: Updates the customized CSS of the user's GUI.
- *currentOnlineUsers*: Returns the list of users currently online in PLE.
- *currentOnlineUserInfo*: Returns some meta data about the online user (session owner).
- *searchForUsers*: Returns the search result for a user in the PLE.

Widget-authentication related services

This API is used for the *RGT* approach as a widget-authentication mechanism by third-party services (see section 3.3.1):

- *getToken*: Returns the user's username and a RGT for a specific subscribed service.
- *check*: Returns if the specified token is valid for the specified user and a subscribed service.

Statistic related services

This API is used by the client to save gathered statistics information on the client in the DBMS on the server (see section 3.4):

- *trace*: Saves the given widget IDs for the online user in the DBMS.

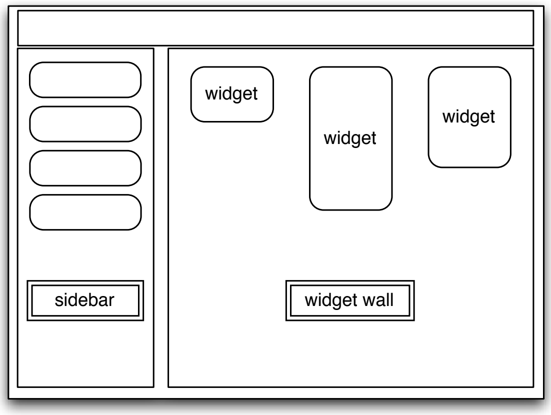


Figure 3.6: The sidebar and widget zone on UI

3.4 Client-Side Architecture

As noted in some of the preceding sections, the PLE is a RIA with a client-server architecture. An AJAX approach is used to transmit data between client and server. The server responds to the client's GET requests with data retrieved from data storage, either in XML or JSON format. The client is responsible for the dynamic creation of the UI and the whole client-side logic.

The client architecture is based on JavaScript. In order to assure browser compatibility on the client side *jQuery*[18], JavaScript library is applied. jQuery is a lightweight JavaScript library that simplifies HTML traversing, event handling, AJAX interactions and everything else related to Dynamic HTML (DHTML) programming (DOM scripting) in the development of RIAs. It supports CSS 1-3 selectors and is a cross-browser solution for web-based client side applications in JavaScript. The GUI is built using the jQuery CSS framework[19] and *jQuery UI*[20].

Section 3.4.1 describes the general structure of the client logic. Section 3.4.2 gives an example of how the initial layout of a widget wall (see section 5.3.2) is dynamically constructed.

As mentioned in section 3.1, the PLE supports event-based communication between widgets. Section 3.4.3 describes how this is rendered possible. As mentioned in the section on use cases (section 3.2.1), a statistics module is implemented in the PLE that gathers statistic data by tracking user behaviour. Section 3.4.4 sheds further light on this issue. The users in PLE are able to change the style sheet of UI in line with their interests. Section 3.4.5 describes this process in detail.

3.4.1 General Structure

The client-side logic is realized with an Object-Oriented (OO) approach in JavaScript programming language. Communication with PLE API is performed through asynchronous XHRs, which results in a higher performance on the client side. Users can work on UI continuously while the client engine communicates with the server (e.g. to load widgets or update user preferences). No interruption would happen in this case.

The client logic is also responsible for the dynamic creation of the GUI. It is composed of several classes that are described below. The relations and dependencies between these classes are illustrated in figure 3.7. Figure 3.6 shows the sidebar and widget zone on the UI that are created dynamically by the corresponding classes.

- ***WidgetFactory***: For each widget that must be loaded to the interface, *WidgetFactory* requests the configuration file from the server API through an XHR, parses the XML content and initializes a

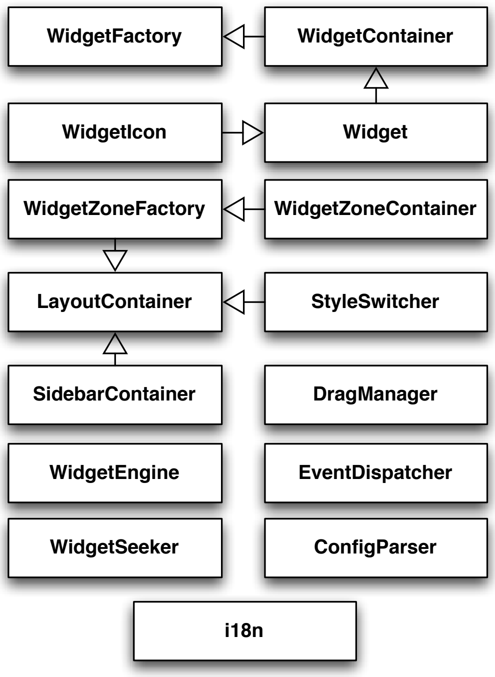


Figure 3.7: The relations and dependencies between classes in client logic

WidgetContainer, which is then saved as a reference.

- ***WidgetContainer*** creates the HTML code of the main widget GUI (toolbar and container) as well as the GUI of the edit window. The widget container is actually an Inline Frame (IFrame) with the *src* referred to the start file of the widget (index.html). *WidgetContainer* brings widgets to life by appending the generated HTML code into the Document Object Model (DOM). Once the HTML code is appended and the widget has been loaded completely, the *widget* object is initialized and injected into the DOM of the widget.

- ***Widget*** contains the methods specified as widget API. The *widget* object is actually the implementation of the extended W3C widgets interface specification described in section 4.3.

- ***WidgetIcon*** contains attributes of a widget icon such as source, width, height, etc.

- ***StyleSwitcher*** contains the functionality to append new CSS to the head of the document, used for switching the style sheet.

- ***SidebarContainer*** initializes and constructs the sidebar elements on the PLE UI. It is also responsible for event handling on sidebar elements and the *Dock menu* functionality.

- ***WidgetZoneContainer*** initializes and constructs widget zones on PLE UI. It is also responsible for event handling on widget zones, such as drag & drop events for widget displacement on the widget wall.

- ***WidgetZoneFactory***: For each widget zone that must be loaded to the interface (depending on categories), *WidgetZoneFactory* initializes a *WidgetZoneContainer*, which is then saved as a reference.

- ***LayoutContainer*** serves as the controller of other classes and is responsible for the main functionalities of the client. It initializes and constructs the whole UI, the widget wall, by retrieving the list of defined categories through an XHR from server API. It contains the unique instances of *SidebarContainer*, *WidgetZoneFactory* and *StyleSwitcher*. It thus controls and triggers the initialization of sidebar, widget zones and the process of style switching. The created GUI (HTML code) for the of sidebar (by *SidebarContainer*) as well as widget zones (by *WidgetZoneFactory*) are appended to the DOM in the initial phase. Furthermore, the interactions between the sidebar and widget zones are managed, user-driven events are set, AJAX related issues are defined and message boxes that may contain alert messages for the user are initialized.

- ***WidgetEngine*** acts as a further controller and contains methods that are used to carry out some of the use cases depicted in figure 3.3. These include ”add widget”, ”load user interface” (widget that a user has already activated), ”update interface” (in case of widget displacement), ”update user preferences”, ”search widget”, etc. As described before, communication and data transmission to the server are performed via asynchronous XHRs.
- ***WidgetSeeker*** is responsible for sending search queries to the server API.
- ***EventDispatcher*** acts as a central listener for events subscribed to by widgets. Refer to section 3.4.3 for more details.
- ***ConfigParser*** is actually an XML parser that parses the content of a widget configuration file and returns the configuration data as a JavaScript multidimensional array.
- ***i18n*** retrieves the translated strings from the server API.

For further information on each class and the list of class members refer to appendix A.2.

3.4.2 Dynamic User Interface (UI) Construction

In the RIAs, as it is the case in PLE, the UI is created dynamically on the client side. This is done with DHTML or DOM scripting. This process is explained here with an example in case of the PLE.

Once a user is authenticated successfully, he is redirected to the widget wall (see section 5.3) where the UI is dynamically constructed using the OO structure described in section 3.4.1. Figure 3.8 depicts the sequence diagram of the dynamic construction of the widget wall for a better illustration.

The following processing steps are performed to build the initial user’s UI on the client side:

- ***LoadLayout()***: The *LayoutContainer*, which is responsible to construct the whole UI, sends a request to the server to receive the categories.
- ***initLayout()*** => ***LoadInterface()***: As soon as the categories arrive on the client side, the *WidgetEngine* is asked to load the user’s interface from the server, which includes all the widgets that the user has already added (activated) in the PLE within different categories (widget walls). The *WidgetEngine* sends a request to the server API to retrieve the widget IDs.
- ***LoadWidgets()***: Afterwards, the *WidgetFactory* is asked to build and load the widgets of the currently active widget wall into the user’s UI. In case of initial UI construction, it would be the ”personal desktop” that appears on top of the sidebar and is loaded by default.
- ***requestManifest()***: *WidgetFactory* requests the manifest file of widgets within the specified widget wall (in this case ”personal desktop”) from the server, by sending the widget IDs that have already been retrieved from the server.
- ***parseManifest()***: Once the manifest files arrive on the client side, they will be parsed and the widget data will be extracted and forwarded to the *WidgetContainer* to build the GUI of the widget (main and edit window) according to the retrieved widget configurations. The *WidgetFactory*

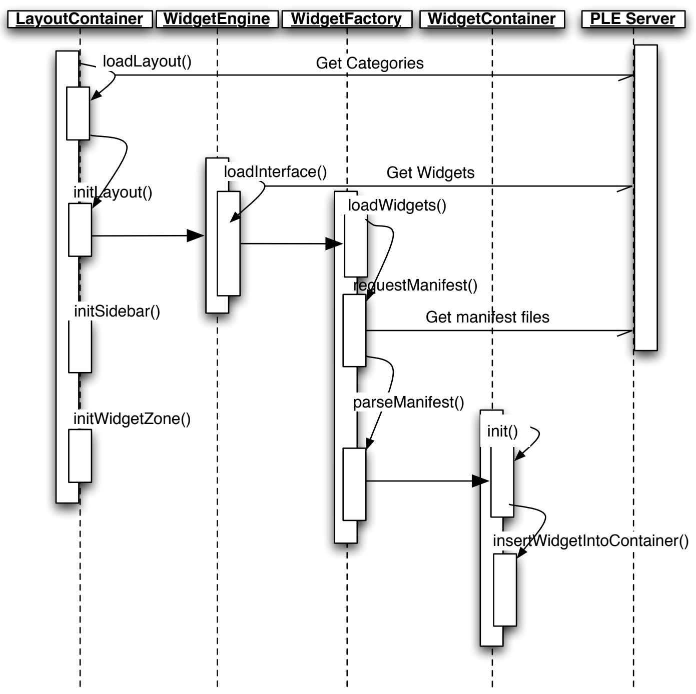


Figure 3.8: The sequence diagram of the dynamic initial UI construction.

contains a reference to each *WidgetContainer* for each widget to access the DOM element of the widget as fast as possible for eventual further processing (e.g. switching to edit window or removing from DOM triggered by a *remove* action).

- ***init()***: *WidgetContainer* initializes the GUI of the widget. At this step, the widget is appended to the DOM and can be viewed by the user in the column and position specified on the widget wall.

- ***insertWidgetIntoContainer()***: As soon as the widget is loaded completely in the PLE (when the DOM of the widget IFrame is ready for access), the *WidgetContainer* creates a new *Widget* object and inserts it into the DOM of the widget. Thus the widget can access user preferences and the *Widget* API to meet user requirements (see section 4.3).

In the meantime, *LayoutContainer* calls *initSidebar()* and *initWidgetZone()* from *SidebarContainer* and *WidgetZoneFactory* to build the GUI related to the sidebar and widget zones on the UI. Since all XHRs in the described process steps above are asynchronous, the client continues to operate and construct the sidebar and widget zones without any interruptions.

3.4.3 Inter-Widget Communication

One of the extensions of W3C specifications is the inter-widget communication mechanism implemented in the Palette project [Naudet et al., 2008]. It allows for an interaction between widgets in the form of unicast, multicast or broadcast communication within the PLE, based on event notifications. Widgets can communicate with each other by sending event messages. The messages, which include the name of an event and the associated data, are transmitted from source widget to target widget (or multiple target widgets) by the widget engine on the client. Through unicast communication, the event and the data are transmitted to only one target widget. Through multicast communication, the message is transmitted to a number of specified target widgets. In the case of broadcast communication, events and data are transmitted to all widgets within the PLE that are listening to that event. The event names must be unique Uniform Resource Identifiers (URIs). The widget engine on the client can also fire events. The widgets can follow these standard events to be notified if an action takes place. The actions can be for instance opening or closing and removing or adding a widget.

The inter-widget communication mechanism is managed in *EventDispatcher*. If a widget adds an event listener, four parameters are saved in a multidimensional array (*listener*) in *EventDispatcher*.

1. The target widget ID as the listener widget that should be notified when an event is fired by any source widgets.
2. The event type, which the target widget listens to. If the specified event is fired, the target widget should be notified.
3. The event handler that is called when the target widget is notified.
4. The accepted source widget IDs as the firing widgets. The target widget should be notified only if any of the specified source widgets fire the specified event type.

If a widget fires an event, the *listener* array is searched for the target widgets that are listening to the fired event and expecting to be notified from the firing source widget(s). If any listening widgets are found, their registered event handlers are easily called.

The inter-widget communication mechanism has been applied in the statistics module that is described in section 3.4.4.

3.4.4 Statistics Module

In order to improve the PLE, it is necessary to consider different parameters that influence the attractiveness and effectiveness of the entire system in general as well as the widgets. To meet this goal, a statistics module has been implemented to measure quantitatively how often widgets are used, and on the other hand to find out more about the quality of user experience. The statistics module provides the system with valuable data that can then be used to analyse user behaviour and widget reputation.

Variation and selection are important mechanisms in the evolutionary development of organismal life forms. These mechanisms were examined and described by Charles Darwin in his famous book [Darwin, 1859]. He argued that there is an advantage in the probability to survive for these individuals and populations, which are able to adapt better to their environment. Following the Darwin model of PLE evolution, this will ensure a stepwise improvement and rejection of rarely used widgets in further iterations of the development cycle.

In order to measure the usage of widgets quantitatively, a hidden module in the background is required to track users' behaviour in relation to widgets. The statistics module is able to collect information on the usage of widgets in detail. The client-side statistics module is added to the PLE widget engine in order to provide widgets, together with the possibility to offer information about user behaviour on the client side. The information (if any) is captured from all activated widgets on users' UI and sent in periodic intervals to the server through the specified API (see section 3.3.4).

The flow of information between the widgets and the statistics module is made possible through event notifications, which are applied in inter-widget communication. For more information on event notifications refer to section 3.4.3.

Each widget should be sufficiently extended to offer information about user behaviour to the statistics module. The information provided by the widget can be detailed. As an example, a Twitter widget can provide the following data about user behaviour:

- TWEETS_READ
- TWEETS_SEND

- DIRECTMESSAGE_SEND
- LIST_CREATE
- FOLLOW

The captured data are then saved in PLE DBMS for later analysis.

3.4.5 Stylability

The GUI of PLE is developed by using a uniform CSS framework by jQuery, since *jQuery UI* is used for a great part of the functionalities in the UI, and *jQuery UI* is based on the jQuery CSS framework. The benefit of using a uniform CSS framework is that the style sheet of the system can be substituted by another one just by using another CSS theme provided by the same framework. In the PLE, this is realized in form of a *preference* widget. In the *preference* widget, the user can select a new theme from a list and then save it as a favourite. In this case, the new selected style sheet is appended as a style sheet link to the head of the document, which would definitely overwrite the rules of the existing ones. In order to be sure that the old style sheets have no effect on the document, they will be disabled.

jQuery has made it easy to generate new themes for the GUIs that base upon the jQuery CSS framework. It provides a web application called *ThemeRoller*[21]. *ThemeRoller* offers an interface for designing and downloading themes for jQuery UI. The number of provided themes in the PLE can be extended easily by using this application. Furthermore, users can also use this application to design their own favourite theme. Using the FireFox bookmarklet, provided by *ThemeRoller*, it is possible to change the theme in the PLE by a single click.

3.5 Widget Development Environment (WDE)

PLE is a mashup of widgets. From the technical point of view, it also acts as a container to deploy and aggregate different widgets within one environment. It plays an important role in providing as many qualitative widgets as possible in the PLE, hence a PLE with few widgets would not meet the expectations of users and appear to be incomplete. This can only be possible if a high number of developers work simultaneously on the development of different widgets.

In the scope of this master thesis, a WDE has been developed to let developers, such as students of informatics who have the know-how on software development, work on widget development. The developed WDE makes widget development independent of the PLE. WDE includes a very light version of the widget engine that is used in the PLE. It contains an `index.html` file as the start file of the environment. Additionally, some JavaScript and CSS files are included in the package for the sake of providing for the functionality and the look & feel of the WDE.

Since the WDE directory contains only HTML, JavaScript and CSS files, it can be started from any browser. The developers need to put the WDE folder somewhere on their local working Personal Computer (PC) and open the start file `index.html` in their favourite browser. The WDE directory also contains an empty subdirectory named `./widgets/`. `./widgets/` will contain the widget folders that the developer will implement.

3.5.1 WDE Restrictions

WDE works relatively well for the development of pure client-side widgets that own no preference definitions in the configuration file.

User Preferences

Since the server-side DBMS is missing in WDE, no preferences can be saved permanently or loaded dynamically. That is why the developer cannot test the widget for different user preferences. A workaround would be to set the new user settings as the value of the attribute `default_value` in the configuration file for the corresponding *preference* (see section 4.2.2). In this way, developers can test the widget for different user preferences.

XHR Restrictions

Since the WDE is normally started from the file system, there is no possibility to work with AJAX requests in a widget. The only solution is to install a local server on the client machine (e.g. Apache[22]) and start the WDE from a local host. This solution enables the sending of XHR to local files that exist in the widget folder.

Since the server-side proxy is missing in WDE, no XHRs can be sent to remote services. The developers are encouraged to use one of the approaches described in section 4.4, otherwise they have to use a separate proxy on their own local host implemented by a server-side dynamic programming language.

In any case, it is not possible to apply the widget authentication mechanism (see section 3.3.1) in WDE, if this is required for widget development.

Chapter 4

Widgets in PLE

" Widgets are full-fledged client-side applications that are authored using Web standards such as [HTML5] and packaged for distribution. They are typically downloaded and installed on a client machine or device where they run as stand-alone applications, but they can also be embedded into Web pages and run in a Web browser. Examples range from simple clocks, stock tickers, news casters, games and weather forecasters, to complex applications that pull data from multiple sources to be "mashed-up" and presented to a user in some interesting and useful way. "

[Widgets Packaging and Configuration 2010]

Widgets (in some applications also known as gadgets) are small client-side applications than run on desktop or a web page. Briefly put, widgets are a portable chunk of code that can be installed and executed on any HTML-based web page or desktop. The code, which is hosted on the client side, can be a simple JavaScript code, a Java Applet or an Adobe Flash code for embedding media players. This code is usually embedded in the <body>-tag of an HTML document on the client side. It includes the client-side programming logic and presentation layer. Widgets allow for a very simple distributed knowledge transfer and diffusion. The code that implements the widget can be located on any server that is accessible through the WWW. As a result, many different distributed services can be provided by the client side, such as a Content Management System (CMS) or a LMS, without any further technical effort [Taraghi et al., 2009c].

What is more, widgets are very often used on personalized web sites, personal desktops or in the PLEs, where users are encouraged and supported to aggregate and create their own configuration of widgets. iGoogle[23], Netvibes[24], Protopage[25] and Pageflakes[26] are some examples of such personalized desktops. The most famous projects that provide developers with tools to develop widgets, are the Konfabulator from Yahoo widgets, Dashboard from the Apple project, Desktop widgets from Opera, and Google gadgets. The disadvantage that these projects have in common is the lack of interoperability for desktop widgets [Taraghi et al., 2009c]. Different types of widgets require different widget engines. Widgets of one widget engine, like iGoogle, cannot be applied in others, like Netvibes. The W3C *widget family of specifications*, which will be introduced in section 4.1 contains a series of specifications to gain a standard for widgets and remove the lack of interoperability among widget engines. The widgets applied in PLE are compatible with the Palette specification, which has been developed within the Palette project [PALETTE, 2008] and extends the W3C widget specifications (Packaging and Configuration as well as API) to some extent. Sections 4.2 and 4.3 describe these extensions in detail.

Due to security restrictions of the XHR object in browsers, client-side applications such as widgets can only send AJAX requests to the URIs on the same domain. If communication is required by widgets to remote servers, a proxy module is applied on the PLE server to make data retrieval from remote servers possible. However, there are some client-side techniques to bypass this restriction without the need to go through a proxy on the same domain. These techniques are summarized in section 4.4.

In the scope of this master thesis, a very simple framework for widget development is developed that bases upon MVC design architecture. The framework has been applied during the development phase of widgets for the PLE and is described in section 4.5.

4.1 The W3C Widgets Family of Specifications

The W3C Widgets Family of Specifications [WidgetSpecs, 2008] includes a set of specifications, which together standardize a widget. The following specifications have been formally published by W3C and are briefly described below to provide a general overview. Other W3C specifications are still not implemented in the PLE widget engine and are just mentioned here for the sake of completeness.

Note: since W3C specifications are updated continuously, some specifications implemented in the PLE widget engine may be outdated. The described packaging and configuration specification in section 4.2 and widget interface in section 4.3 correspond to the actual version of the PLE widget engine.

4.1.1 Widgets Packaging and Configuration

According to Widgets Packaging and Configuration[27], ”widgets are client-side applications that are authored using Web standards such as HTML 5, but whose content can also be embedded into Web documents.”

The Widgets Packaging and Configuration specification standardizes a .zip packaging format, which includes the whole widget source code with a specified file structure and an XML-based configuration file with some mandatory and non-mandatory elements that declare meta data and configuration parameters for a widget. The packaging format acts as a container for files used by a widget and includes some obligatory and non-obligatory elements. It also describes how internationalization and localization must be applied within the packaging format. Moreover, this specification determines a series of steps that should (must) be followed by developers while they implement widgets. Finally, the behaviour and the means of error handling for widget user agents are also specified.

The PLE widget folder structure and XML-based configuration file are described in detail along with Palette extensions in section 4.2.

4.1.2 Widgets Interface

The Widgets Interface specification [28] defines an API for the functionality of widgets. It describes the means to access the data defined in a widget's configuration file, such as the widget's metadata and widget-related user preferences, which are stored as persistent data. It can furthermore be used to handle events related to the changes in the view state of a widget. It also determines the locale under which a widget is currently running (to support localisation), and the notification mechanism of events relating to the widget being updated. Other points in this specification are how to invoke a widget to open a URL on the user's default browser and how to request the user's attention to the widget in a device-independent manner.

The widget interface and the additional extension that are applied in PLE are described in detail in section 4.3.

4.1.3 Widgets Digital Signature

Widgets Digital Signature [29] is another W3C widget specification that deals with the digital signing of a widget package if required, by using a custom profile of the XML-Signature Syntax and Processing Specification. It gives a description of signature syntax and its usage in widget packages along with an explanation on their generation and validation. Next to that, signature algorithms and processing rules for widget user agents are described as well.

4.1.4 Widget Updates over HTTP

This is a specification [30] that defines a version control model which enables widgets to remain updated over HTTP. It defines a process and a document format to enable users to update an already installed widget package with a different version of a widget package via HTTP and via non-HTTP sources (for instance directly from the hard disk or memory card of the device).

4.1.5 Widget Access Request Policy

The Widget Access Request Policy specification [31] describes a method to obtain access to web resources. In the scope of this specification, the security model is defined as a method for widget authors to file a request to the user agent to grant them access to specific network resources.

4.1.6 Widgets 1.0: URI Scheme

Widgets 1.0: URI Scheme specification [32] defines a URI scheme that is used to address resources inside a widget package. Among others, the syntax, the authoring base URI, the relative URI reference resolution and mapping widget URIs to files incorporated in a widget package are presented.

4.1.7 Widgets 'view-mode' Media Feature

The 'view-mode' Media Feature specification [33] defines a CSS Media Query in relation to the mode of presentation of a document's contents.

4.1.8 Widgets 1.0: Requirements

This document [34] enumerates the design aims and requirements that specifications ought to address in order to standardize different aspects of widgets.

4.1.9 Widgets 1.0: The Widget Landscape

In this document [35], a survey is performed among a group of market-leading widget user agents in order to inform the requirements of Widgets 1.0: Requirements document.

4.2 Packaging and Configuration

The instructions for widget packaging and configuration apply only to local widgets in the PLE. The .zip archive file format, defined in the ZIP specification [36], is the packaging format for local widget packages.

In case of remote widgets, the folder containing widget files must be accessible over HTTP and can have any kind of structure. The start file of the widget must be a valid index file according to the remote server configuration and must output a valid (X)HTML.

4.2.1 Folder Structure

The widget folder (.zip file) **must** contain the two following files at its root [Naudet et al., 2008]:

- index.html: This is the start file that is displayed in the browser while the widget is loaded. Normally it contains the whole JavaScript functionality, CSS and HTML structure of the widget.
- config.xml: The configuration or manifest file is an XML document that contains all the metadata needed to initialize and run widgets in a PLE, such as a widget's name, identity, width, height, description, author, icon reference, preferences, etc. For more information about the manifest file refer to section 4.2.2.

The widget folder **may** contain some non-mandatory files and folders that are necessary to meet widget requirements, such as JavaScript and CSS files, images, etc.

4.2.2 Configuration File

The configuration file is an XML document, which contains elements describing the widget and its preferences, which can be customized by the user. The widget configuration file, which is specified in W3C widget packaging and configuration, has been extended to add some default user preference values in order to facilitate widget customization.

The configuration file is extended in the Palette project [Naudet et al., 2008], where two different namespaces are used; one namespace for default W3C specifications (`http://www.w3.org/TR/widgets`) and the other for the Palette specification (`http://palette.ercim.ord/ns/`). All extensions **must** be specified in the *palette* namespace.

The root element of the configuration file is the widget element that **must** be present. It **must** contain the following three attributes:

- id: The unique identifier to identify the widget within the PLE environment.
- width: The width of the widget in pixel
- height: The height of the widget in pixel

Note: The width is actually ignored in the PLE and must be specified, thus it must be present according to the XSD. In the PLE UI the width is set automatically to exploit the whole width of the user screen.

The widget element contains the following mandatory element as child:

- title: A human-readable title for the widget that is displayed to the user.

A simple configuration File can be viewed in Listing 4.1.

```
<widget
  xmlns=http://www.w3.org/TR/widgets/
  xmlns:palette=http://palette.ercim.org/ns/
  id="helloWorldWidget" width="200" height="300">
  <title>Hello World!</title>
  <icon src="icon.png" width="16" height="16" />
  <author url="http://mmustermann.com" email="mm@example.org">
    Mustermann
  </author>
  <description>Widget Description</description>
</widget>
```

Listing 4.1: A simple configuration file

The widget element contains the following non-mandatory elements as children:

- author: The name of the widget developer. It may contain email and url as attributes.
- description: The description of the widget in plain text.
- license: The license of the widget in plain text.
- version: The version of the widget in plain text.
- icon: The path to the widget icon. It may contain width and height as attributes.
- scrollable: It can contain a boolean value to make the widget content scrollable.
- alternative_url: The URL to an alternative view for the widget. For instance in case of the Google Translator widget, it can be the URL to the Google translation service that is provided by the widget. It must be specified in the *palette* namespace.
- widget_type: It can contain one of the values "*local*" or "*remote*" and can be used to define the widget as a remote one. It must be specified in the *palette* namespace.
- widget_location: It applies only to remote widgets and is ignored in local widgets. In case the widget_type element is set to "*remote*", this element **must** be present. It contains the URI to the folder on the web, in which the widget index file is expected. It must be specified in the *palette* namespace.
- widget_authentication: It can contain one of the values "*enabled*" or "*disabled*" and can be used to turn on/off the widget authentication mechanism (see section 4.3.4).
- preferences: Through the preferences element, customizable user preferences can be defined. The preferences are stored continuously on the PLE server and are accessible to the widget via its interface (see section 4.3). It must be specified in the *palette* namespace.
  This element contains a preference child element for each user preference. An example of defined user preferences can be viewed in Listing 4.2.
- preference: Every customizable user preference must be defined as preference element.
  preference elements **must** be children of the preferences element and must be specified in the *palette* namespace. The attributes of the preference element are as follows:
  - name: The name of the variable, under which the preference data is perststistently stored. It **must** be present.

  - display_name: Optional string to display to the user on the edit window.
  - default_value: Optional string to be set as default value for the preference element.
  - datatype: Optional string to specify the type of stored data. Possible types are: string (default), number, bool, hidden and enumeration.
    The enumeration data type is used to provide the user with the possibility to select a user preference out of different choices. The choices must be defined as enumeration elements as children of the preference element. If the preference element is of type enumeration, the enumeration elements must exist. An example of enumerations can be viewed in Listing 4.3.

- enumeration: This element **must** be a child of the preference element and must be specified in the *palette* namespace. The attributes of the preference element are:
  - value: The value of a user preference that is stored persistently if selected by the user. It **must** be present.
  - display_value: Optional string to display to the user in the edit window.

```
<palette:preferences>
  <palette:preference name="address" display_name="Address"
    datatype="string"/>
  <palette:preference name="tel" display_name="Tel." datatype="number"/>
</palette:preferences>
```

Listing 4.2: Preferences defined in configuration file

```
<palette:preferences>
<palette:preference name="lang" display_name="Language"
  datatype="enumeration" default_value="de">
  <palette:enumeration value="de" display_value="German"/>
  <palette:enumeration value="en" display_value="English"/>
  <palette:enumeration value="es" display_value="Spanish"/>
 </palette:preference>
</palette:preferences>
```

Listing 4.3: Preferences defined as enumerations

For more information about the XSD of the configuration file refer to appendix A.3 [Naudet et al., 2008].

4.3 Application Programming Interface (API)

The W3C widget interface has been extended to enable widget intercommunication within the PLE environment. Communication can run in the background automatically or can be directed manually by the user, for instance as a drag and drop event for data flow between two widgets [Naudet et al., 2008].

The widget API is accessible in JavaScript through the widget object. The widget object is inserted into the DOM of the widget index file as soon as the widget is completely loaded in the browser to expose the functionality to the widget. Once the widget is fully loaded, the PLE calls the onLoad() function, which may be defined in the widget. onLoad() function should replace the onload() event specified in HTML 4 and serve as the start of a widget functionality. Widget developers can be sure that the widget object is available within the onLoad() function and they can access the widget API.
Note: The widget object is not available in JavaScript for remote widgets. As a result, remote widgets cannot use the methods specified in the widget interface.

4.3.1 Read and Write User Preference Data

The following two methods provide the functionality to access and store preference data on the PLE server [Naudet et al., 2008]. *Key* stands for the variable name that is defined for preference data in the configuration file.

- *<datatype> preferenceForKey (string key)*
- *void setPreferenceForKey (<datatype> preference, string key)*

For remote widgets, user preferences are appended to the widget URL as GET parameters (key=preference). It is possible to update preferences only through the built-in client logic in the PLE edit window. In other words, since the widget interface is not available for remote widgets, they can only be customized by users through the PLE user interface.

4.3.2 XMLHttpRequest Methods

The widget interface provides five built-in XHR methods to retrieve data from local or remote servers [Naudet et al., 2008].

1. *void httpGet (string URI, object params, function callback, function errorCallback)*
   This method sends an asynchronous HTTP *GET* request to the specified URI with the parameters *params*. The response is intelligently parsed as either responseText or responseXML.

2. *void httpGetJSON (string URI, object params, function callback), function errorCallback)*
   This method sends an asynchronous HTTP *GET* request to the specified URI with the parameters *params*. The response is parsed as JSON.

3. *void httpPost (string URI, object params, function callback, [string contentType], function errorCallback)*
   This method sends an asynchronous HTTP *POST* request to the specified URI with the parameters *params*. The response is intelligently parsed as either responseText or responseXML.

4. *void httpPut (string URI, object params, function callback), [string contentType])*
   This method sends an asynchronous HTTP *PUT* request to the specified URI with the parameters *params*. The response is intelligently parsed as either responseText or responseXML.

5. *void httpDelete (string URI, object params, function callback)*
   This method sends an asynchronous HTTP *DELETE* request to the specified URI with the parameters *params*. The response is intelligently parsed as either responseText or responseXML.

The *params* are key value pairs in JavaScript. In case the HTTP response code is 200, the *callback* function is called, otherwise the *errorCallback* function is triggered. The actual response from the server is passed on as the first argument to the *callback* function.

Because of security restrictions of the XHR object imposed by browsers, which only allows to access files on the same domain, a proxy module is applied on the PLE server. In case of requests by a remote server, the XHR methods send requests directly to the proxy module on the PLE server. The proxy acts as a bridge and passes all requests and replies unmodified from the client to the remote server and vice versa (see section 3.3.2).

There are some techniques to bypass the restriction of the XHR object in browsers, which are described in section 4.4. If these techniques were applied in widget development, there would be no need to use XHR API.

4.3.3 Read and Write Widget Settings

The following methods provide the possibility to access the metadata defined in the configuration file and readjust some features of the widget if desired:

- *<number> getDefaultHeight ()*
  returns the default widget height set in configuration file.
- *<string> getDefaultTitle ()*
  returns the default widget title set in configuration file.
- *<number> getHeight ()*
  returns the height of the widget.
- *<string> getTitle ()*
  returns the title of the widget.
- *void openURL (<string> URI)*
  opens the specified URI in a new window
- *void setHeight (<number> height)*
  sets the height of the widget
- *void setTitle (<string> title)*
  sets the title of the widget
- *void setContentProxy (<string> URI)*
  sets the specified URI as the path to proxy module on PLE server

4.3.4 Widget Authentication

The following methods [Naudet et al., 2008] deal with the normal HTTP authentication and widget authentication mechanism for the case the widget needs to be authenticated by third-party services, mainly by remote services (refer to section 3.3.1 for more information).

- *void setHttpCredentials (<string> usr, <string> pwd)*
  sets the specified username and password in the header of XHR to be applied for HTTP authentication.
- *void enableAuthentication ()*
  enables the widget authentication mechanism.

4.3.5 Inter-Widget Communication

One of the extensions to W3C specifications is the inter-widget communication mechanism implemented in the Palette project [Naudet et al., 2008]. Widgets can add listeners to events or fire events to notify other widgets. The interaction between widgets can be unicast (widget to widget), multicast (widget to more than one widget) or broadcast (widget to all widgets) communication.

The following methods describe the messaging API, which provides the possibility for widgets to add listeners to events, remove events or fire some events in one of the types of communication listed above.

- *void addWidgetEventListener (<string> eventType, <string> eventHandler , <string> acceptedSource)*
  listens to an event of the specified event type with the specified event handler from specified source widget(s). When an event is fired, the function eventHandler is executed. If accepted-Source is null, the widget will be notified for event of type eventType from any source widget.
- *void removeWidgetEventListener (<string> eventType, <string> source)*
  stops listening to the specified event type from the specified source widget(s). If source is null, the widget stops listening to eventType events.
- *void fireWidgetEvent (<string> target, <string> eventType, <datatype> data)*
  fires an event of the specified event type to one or several target widgets with the attached data. If target is null, it would be a broadcast event notification.

Manual inter-widget communication by Drag & Drop

While inter-widget communication can be run automatically in the background, it can also be triggered manually by the user [Naudet et al., 2008]. For this case, two more methods are added to the messaging API to make manual communication between two widgets (unicast) possible:

- *bindWidgetToDropType (<string> eventType)*
  defines the widget as a drop target for drag & drop events of the specified eventType. The widget is notified if a dragged widget element is dropped on it.
- *addDragData (<DOM element> target, <string> eventType, <datatype> data, <string> tooltip)*
  manipulates the specified DOM element of the widget into a draggable element, triggers a drag event of the type eventType with associated data and displays a tooltip message during dragging.

In future, this API can be substituted by the HTML 5 cross-document drag & drop feature.

4.4 Cross-Domain XMLHttpRequest (XHR)

The widgets that represent a remote service on the WWW need a way to send XHRs to APIs on remote domains. However, browsers do not allow such requests to be sent to remote domains due to security restrictions. Although there is a built-in proxy on the PLE server that can be used for this purpose, there are some client-side techniques that can be applied in widget development to communicate directly to remote servers without the need to go via the server-side proxy. These techniques are investigated in the scope of this master thesis and are described here for the sake of completeness.

4.4.1 JSONP

JSON with Padding (JSONP) is an approach to carry out cross-domain AJAX requests. It can be realized by appending a <script/> tag dynamically to the header of the HTML document when the sending of a cross-site request is required. The *src* value of the <script/> tag corresponds to the URI of JSONP API on the remote server. A callback function name must be added as a GET parameter to the URI of JSONP API. Through this approach, the widget actually loads a JavaScript code from the remote server. The JavaScript code contains merely a JavaScript function definition with the name specified as the GET parameter to the URI of JSONP API. The requested data is the return value of the retrieved JavaScript function and is in JSON encoded format. Just after the JavaScript code is downloaded, calling the specified function would give back the desired data resources.

This approach can only be applied if the remote service provides the clients with a corresponding JSONP API.

4.4.2 YQL Proxy

Yahoo! Query Language (YQL)[37] is a SQL-like language that lets the developers query, filter and join data across Web services through Yahoo API. This technique resembles the JSONP approach to some extent. In both techniques, the API is called through the HTML <script/> tag. The difference is that instead of calling the API of the remote service directly, the Yahoo API is called and used as a proxy to fetch data from different Web services.

Yahoo API has a cache mechanism for retrieved data that seems to be very fast. On the other hand, the developers can use the SQL-like language to query, filter, and join data across Web services. The SQL-like query (SELECT, UPDATE, INSERT and DELETE) can be set as the value of the GET parameter q (URI-encoded) to the URI of the Yahoo API. When the YQL processes a query, it retrieves a data resource on the WWW, transforms the data, and returns the results in either XML or JSON format. YQL can retrieve several data resources such as Yahoo and other Web Services as well as Web content in HTML, XML, Really Simple Syndication (RSS), and Atom formats. Some other configurations can be set by GET parameters in URI as well. For instance, it is possible to specify the return format of data retrieved from Web services. A callback function name can be specified to be called in order to handle the requested data on the client side. More information related to YQL can be found on the official YQL web site (`http://developer.yahoo.com/yql/`).

4.4.3 CSSHttpRequest

CSSHttpRequest[38] is cross-domain AJAX through CSS. It functions similarly to JSONP, but using CSS as CSS is not subject to the Same-Origin Policy (SOP) that affects XHR. The data is encoded on the server into URI-encoded 2KB chunks and serialized into CSS rules with a modified *data:* URI scheme[39]. For more information about *CSSHttpRequest* refer to the official web site (`http://nb.io/hacks/csshttprequest`).

This approach can be only applied if the remote service supports *CSSHttpRequest*.

4.4.4 Flash Proxy

Cross-Domain requests can be processed with Flash as a client-side proxy. *CrossXhr*[40] and *flXHR*[41] are two tools hat follow this approach to realize a Flash-based client-side proxy. Cross-Domain requests with Flash must be permitted by the remote domain. This is done by a file called crossdomain.xml, which must exist on the remote server. The service provider can limit access to its services and allow only specific domains to call its services by defining the domain names in the XML file mentioned above. If third-party services forbid the request or if they do not have any crossdomain.xml file, this technique is useless. On the other hand, this approach requires Flash to be installed in the browser. A 1-pixel transparent Flash is embedded in the top-left corner of the page as a replacement for the XHR object in browsers.

4.4.5 IFrames

IFrames can be applied to request data from remote servers. As an example, a client from domain A requires to retrieve data resources from a service on domain B. In this case, the client calls (opens) the remote service in an IFrame (iB). The IFrame content, which is actually loaded from remote domain B, must contain an IFrame (iA) referencing to a dynamic script on domain A. The service sets the response data serialized as GET parameter in the *src* of the IFrame iA. Now the dynamic script on domain A, which is called through the IFrame iA, receives the data through the GET parameter and writes them down in a cookie on the client side. The client must wait until the cookie is set. Then the client reads the data from the cookie and destroys it at the end.

This is a very primitive approach. It has many security leaks and cannot be used for large data. In addition to that, the remote script must use the technique described above.

4.4.6 HTTP Access Controls

Cross-site access control is a way for web servers to enable secure cross-site data transfers. The Web Applications Working Group[42] within the W3C has proposed the Cross-Origin Resource Sharing (CORS)[43] to meet this goal. On the client side, the browser handles the components of cross-origin sharing, including headers and policy enforcement. This capability means that the servers have to handle new headers and send resources back with new headers. Similar to *CrossXhr*, the server can restrict access to specific domains. This, however, is done in response headers. Currently Firefox 3.5+, Safari 4 and Google Chrome 2 have implemented the CORS using XMLHttpRequest object. Internet Explorer (IE) 8 implements parts of the CORS specification, using *XDomainRequest* object[44]. A complete treatment of CORS and the XMLHttpRequest object can be found on the Mozilla Developer Wiki[45] (`https://developer.mozilla.org/En/HTTP_access_control`).

4.5 Simple MVC Framework for Widget Development

The great advantage of a RIA is the improved performance, since a great part of the processing logic can be performed on the client-side rather than the server-side. For many web-based RIAs, JavaScript is the most common programming language as it is popular as a script language for browsers. While server-side programming languages have the advantages of the OO programming paradigm, JavaScript bases upon objects with a specific object literal notation. Using design architectures such as MVC in JavaScript, reduces the code complexity and allows for a semi-parallel application development. MVC design architecture was first published in Smalltalk-80 by Glenn Krasner and Stephen Pope [Krasner and Pope, 1988]. It enables an easier and much less time-consuming development on further extensions of RIAs. Web-based widgets as such, developed in the PLE environment, are actually RIAs. In the scope of this master thesis, a very simple framework for widget development is developed that bases upon MVC design architecture [Taraghi and Ebner, 2010]. The framework has been applied during the development phase of widgets for the PLE and appeared to be mostly appropriate for students and RIA developers who have beginner knowledge/experience with JavaScript programming language.

Especially in interactive environments like PLE, user requirements and desires change very often. Therefore, each widget must be easily extendible in the sense of the features it provides and its adaptation to new technologies. The designed framework is suitable to meet these requirements. It can guarantee greater scalability and less complexity in widget development. It can also be applied in any other RIA developed in JavaScript. The biggest advantage of the framework, in comparison with other conventional frameworks, is its simplicity and resemblance to OO programming in server-side languages. Especially for developers who are not advanced JavaScript programmers or have beginner knowledge/experience with client-side programming, but are familiar with OO paradigm, this framework is an appropriate tool to start with.

Section 4.5.1 describes MVC design architecture in general and mentions some existing MVC frameworks implemented in JavaScript programming language. Section 4.5.2 goes over the simple MVC framework that is developed in the scope of this master thesis for easier and more efficient widget development. Last but not least, section 4.5.3 describes in detail how the designed framework can be used and expanded to different submodules.

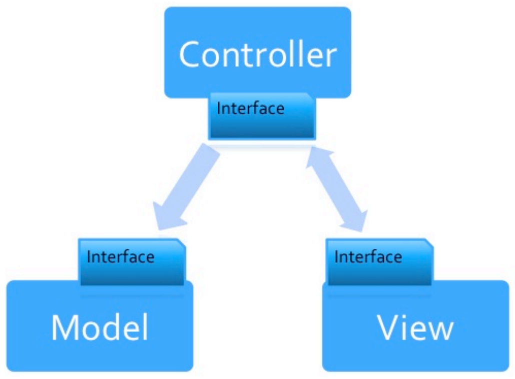


Figure 4.1: MVC Design Structure

4.5.1 MVC Frameworks

The great advantage of a RIA is an improved performance, since a great part of the processing can be done on the client-side rather than the server-side. For many web-based RIAs, JavaScript is the most common programming language because of its popularity as a script language for browsers. While server-side programming languages have the advantages of the OO programming paradigm, JavaScript bases upon objects with a specific object literal notation. There are several different ways to implement inheritance and class-like structures in JavaScript as described in [Crockford, 2008]. By using design architectures, such as MVC in JavaScript, the code complexity is reduced and it allows for semi-parallel application development.

MVC design architecture bases upon the separation of three main distinctive components of an application:

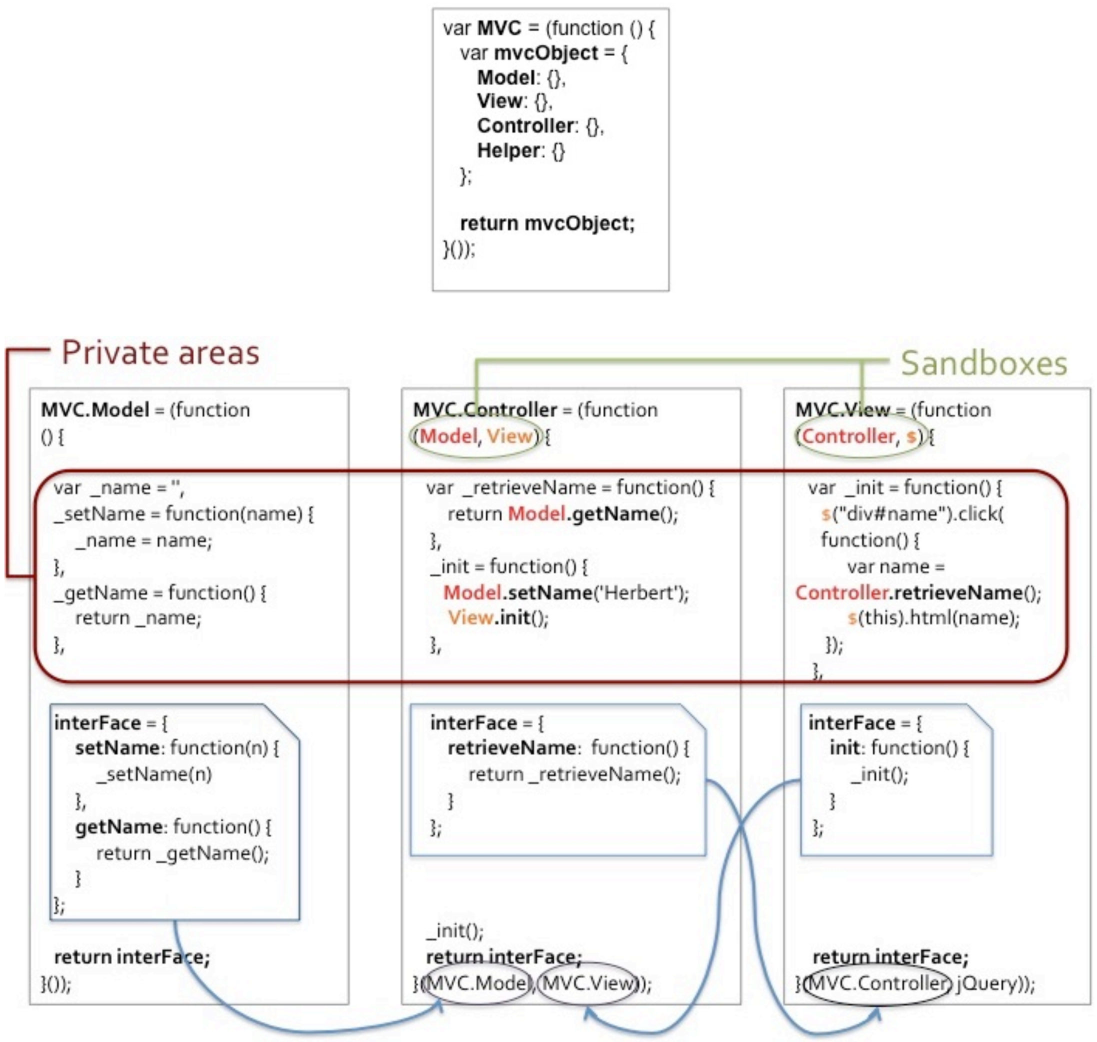
- The presentation layer (view) that deals with the visual display of the application and the whole GUI.
- The logic layer (controller) that describes the behaviour of the application and connecting components, presentation and data layer to each other.
- The data layer (model) that is responsible for data collected on the client-side.

Figure 4.1 displays the structure of an MVC framework.

There are already some works in the field of MVC design architecture for JavaScript. JavaScriptMVC[46] is one of the open-source frameworks. In addition to MVC architecture, it provides features such as concatenation, compression, testing modules, error reporting, etc. TrimJunction[47] is a clone of the Ruby On Rails[48] web MVC framework for JavaScript. PureMVC[49] provides a lightweight implementation of MVC design architecture in different programming languages, including JavaScript. Sproutcore[50] is an HTML 5 application framework for building responsive client applications in modern browsers and bases upon MVC design architecture.

4.5.2 The Simple MVC Framework

PLE is actually a widget container. It would have no benefits without widgets. The more various widgets exist in PLE, the more interesting the environment becomes for the users. For example the students of informatics at TU Graz, who have the necessary know-how in software development, help to implement many various widgets, as shown in [Ebner and Taraghi, 2010]. They are mostly experienced programmers in server-side programming languages, that is why they know the OO paradigm very well and are familiar with different MVC frameworks in server-side programming languages such as Java or



```
var MVC = (function () {
   var mvcObject = {
      Model: {},
      View: {},
      Controller: {},
      Helper: {}
   };

   return mvcObject;
}());
```

```
MVC.Model = (function
() {

  var _name = '',
  _setName = function(name) {
     _name = name;
  },
  _getName = function() {
     return _name;
  },

  interFace = {
     setName: function(n) {
        _setName(n)
     },
     getName: function() {
        return _getName();
     }
  };

  return interFace;
}());
```

```
MVC.Controller = (function
(Model, View){

  var _retrieveName = function() {
     return Model.getName();
  },
  _init = function() {
     Model.setName('Herbert');
     View.init();
  },

  interFace = {
    retrieveName: function() {
       return _retrieveName();
    }
  };

  _init();
  return interFace;
}(MVC.Model, MVC.View));
```

```
MVC.View = (function
(Controller, $){

  var _init = function() {
    $("div#name").click(
    function() {
       var name =
  Controller.retrieveName();
       $(this).html(name);
    });
  },

  interFace = {
    init: function() {
       _init();
    }
  };

  return interFace;
}(MVC.Controller, jQuery));
```

Figure 4.2: Simple MVC Framework in JavaScript used for developing web-based widgets

PHP. On the other hand, not all of them have sufficient experience in client-side programming, at least not such as would be necessary for JavaScript. The conventional MVC frameworks provide (advanced) developers with features that are beneficial for implementing advanced RIAs. The support for these features increases the complexity of the frameworks, which is extremely time-consuming to learn and hence a disadvantage for beginners or less experienced JavaScript developers. In order to make widget development for students as simple as possible, the plan was to apply a very simple and lightweight MVC architecture that aims to become very easy to start with. It is realized through a module pattern. Each three components of MVC are actually a module in JavaScript. Modules use a singleton paradigm, support private data and stay out of the global namespace. These features suit the module pattern to be applied for the model, view and controller in projects that may grow in the course of time and become more complex. The modules themselves are implemented by using closure functions in JavaScript. In Douglas Crockford's book [Crockford, 2008] modules, closures and the singleton paradigm are very well described.

Less experienced students, who are actually advanced OO developers in server-side programming languages, need to consider the modules as three static classes with the ability to contain private and public methods or variables. They can then start implementing the three modules as they are used to in server-side OO programming languages. They can extend the framework by adding additional submodules to support features that are provided by conventional MVC frameworks such as templates, event handling, etc.

Figure 4.2 demonstrates a very simple example of the MVC architecture based on the described structure above for deeper comprehension.

The outlined private area in figure 4.2 contains private members that are valid only within the scope of the corresponding module. The interFace objects contain public members of the corresponding modules. Public members can get access to private members as expected. The view and model modules are in the sandbox of the controller (passed on as arguments of the closure function: MVC.Model and MVC.View). Consequently, the controller can call public members of the view and model interfaces through the local objects Model and View. Correspondingly, the view can access public members of the controller for event handling and the methods of jQuery framework for DOM manipulation.

It is possible to define some default public members in each module directly in the definition of MVC closure. If a module already contains such default members, its interFace object must not be initialized, but extended in order to avoid unsetting the default-defined members. An example of such a case can be seen in listing 4.4. The interFace object of the View module defined in that example is extended, so that the default members, if any are set, are maintained.

In the case of widgets, we need a special type of functionality, such as distribution of user requirements across multiple pages and navigation among pages in UI. This could be realized e.g. with the help of a *page* module as an additional module, or a submodule in View. Submodules can be used as helpers in all three main modules and have exactly the same structure and benefits as the main modules.

With the help of such architecture a very simple framework is realized, which resembles the classes and their public and private members in server-side programming languages. Widget developers have the possibility to control whether (sub)module A is allowed to access (sub)module B by setting (sub)module B in the sandbox of (sub)module A. Through this approach, all (sub)modules and their members stay out of the global namespace; the code is readable, maintainable and easily extendible.

4.5.3 Extendibility Examples

As mentioned above, through the use of a module pattern the MVC architecture is easily extendible. Widget developers can use their experience from server-side programming to add additional (sub)modules and refine the JavaScript code in detail. Here, some extensions in View and Model are considered. The examples are kept very simple and basic for a better insight.

View Extension Examples

As an example of extension possibilities, we can consider the View module and extend it by using templates. View is the actual presentation layer in the MVC design architecture. The public methods in View are normally called by the Controller to display the retrieved data in the UI. In the example shown in figure 4.2, View.init() is called. When the user clicks on an HTML <div> element, some data (here the name) are retrieved by the Controller and put into a node in DOM.

Templates can be used to reduce the complexity of View and separate the HTML layout from the actual View's functionality. Web designers are able to create the templates independently of the programmers who are responsible for three main units (Model, View and Controller). Listing 4.4 shows this extension in a very simple style for a deeper insight.

```
MVC.View.Templates = (function(interFace) {
    var _innerHtml = '';

    interFace.createPersonProfile = function(P) {
        _innerHtml = '<p>Name: <span>'+P.firstname+'</span></p>';
        _innerHtml += '<p>Last name: <span>'+P.lastname+'</span></p>';
        _innerHtml += '<p>Tel: <span>'+P.tel+'</span></p>';
        _innerHtml += '<p>Email: <span>'+P.email+'</span></p>';
        return _innerHtml;
    }
    return interFace;
}({}));

MVC.View = (function (interFace, Controller, $, Templates) {
    interFace.init = function() {
        $(document).ready(function() {
            _init();
        });
    };

    /* private methods */
    var _init = function() {
        $("button").click(function() {
            var P = Controller.retrieveProfile();
            var html = Templates.createPersonProfile(P)
            $(this).next().html(html);
        });
    };
    return interFace;
}(MVC.View || {}, MVC.Controller, jQuery, MVC.View.Templates));
```

Listing 4.4: Extension of the *View* to use HTML templates by adding a new submodule *Templates*

Different templates can be put into one submodule for View. For the sake of better understanding, this example is kept as simple as possible. In real applications, different template engines can be applied, such as JavaScript Templates (JST)[51], PURE JavaScript Template Engine[52], Closure Templates[53] and jQuery template plugin[54].

Model Extension Examples

As another example, Model can be extended. Model acts as the data layer in the MVC framework and can be responsible for data retrieval from data resources (from the same domain or remote servers) through XHRs. Creating a submodule for XHRs can have certain benefits. Different XHR submodules (AJAX, cross-domain requests, etc.) can stand for data retrieval from a remote server in Model. The submodule can be configured for different widget specifications. For instance, in the first case the developed widget is applied as a gadget in iGoogle and therefore the widget must be adapted correspondingly. In this case, no total refactoring is needed in Model. The XHR submodule must only be extended to use proxies specified for iGoogle gadgets. For W3C widgets and other specifications, the same approach must be applied.

Another example is the use of one of the HTML 5 features for caching in Model. Caching is an efficient approach to increase performance in RIAs. Local storage can be used to cache data on browser side. This approach can be applied for instance by using jStorage[55] in a submodule. jStorage is a simple plugin for Prototype[56], MooTools[57] and jQuery to cache data (string, numbers, objects, even XML nodes) on the browser side, making use of HTML 5 local storage.

Listing 4.5 shows model extensions with XHR and HTML 5 local storage. In this example, the data is retrieved from remote resources through the XHR submodule and saved on local storage if the data has not already been saved before. Otherwise the saved data is returned directly from local storage. The XHR submodule is extended to support the W3C widget specification. In the case of a W3C widget, the specified proxy is used. Otherwise (i.e. for the case of a normal web-based widget) it is assumed that the remote service is on the same domain and therefore a normal AJAX request is sent.

```
MVC.Model.XHR = (function ($, spec) {
    var interFace = {
        fetchCourses: function(url, user_id, callback){
            _sendRequest(url, {'L': user_id}, callback);
        }
    },

    _sendRequest = function(service_url, param, callback){
        switch(spec) {
            case 'W3C':
                proxy.send(service_url, param, callback);
            break;
            default:
                $.getJSON(service_url, param, callback);
            break;
        }
    };
    return interFace;
}(jQuery, Config.specification));

MVC.Model = (function (interFace, $, XHR) {
    interFace.getListOfCourses = function(addr) {
        return _getCourses(addr);
    };
    /* private methods */
    var requestIsActive = false,
    _getCourses = function(addr) {
        var value = $.jStorage.get(addr);
        if(!value){
            // if not - load the data from the server
            if(!requestIsActive) {
                requestIsActive = true;
                XHR.fetchCourses ('http://example/service', addr,
                function(data) {
                    value = data;
                    // and save it
                    $.jStorage.set(addr,value);
                    requestIsActive = false;
                });
            }
            return _getCourses(addr);
        }
        else return value;
    };
    return interFace;
}(MVC.Model || {}, jQuery, MVC.Model.XHR));
```

Listing 4.5: Extension of the *Model* to make use of a separate XHR submodule and HTML 5 local storage. The XHR submodule supports data retrieval for W3C widgets.

Chapter 5

PLE First Prototype

" Some of us will do our jobs well and some will not, but we will be judged by only one thing - the result. "

[Vince Lombardi.]

The implemented first prototype of PLE offers centralized access to various university services [Ebner et al., 2010], among others to administration systems, such as TUGraz online[58], LMS: TUGTC[59], or blogospheres: TU Graz LearnLand (TUGLL)[60] [Ebner and Taraghi, 2008] in one overview. The users can personalize their PLE to their individual information and learning needs. Currently only one widget is provided for each university service; however, for each use case a widget will be developed to cover all relevant services in the PLE. For the time being, for searching in TUGraz online and browsing through courses in TUGTC, a blog reader for TUGLL, e-mail and newsgroups widgets are integrated. As an example, TUGLL provides many other services [Ebner et al., 2008], such as social bookmarking, file sharing, semantically enriched tag search [Softic et al., 2009], etc., that can be provided in a PLE through widgets.

What is more, public services on the WWW are also offered in the PLE. For each of these services, a widget has been developed that can be integrated into the PLE. Figure 5.1 shows a conceptual view of the first prototype of PLE that integrates university portals as well as other Internet services.

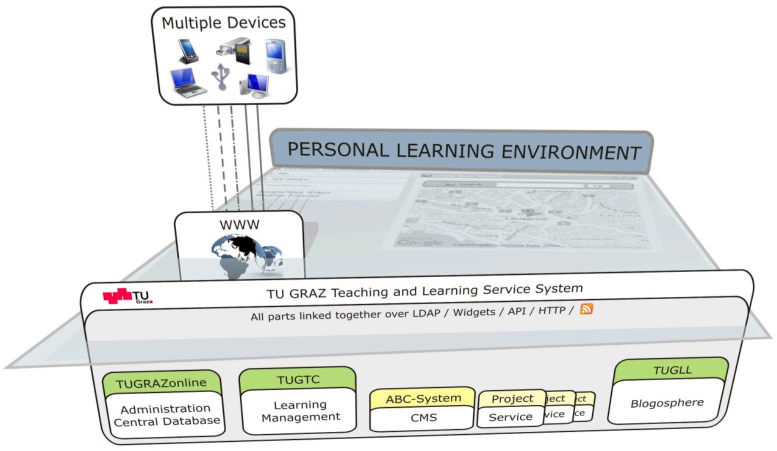

Multiple Devices
PERSONAL LEARNING ENVIRONMENT
WWW
TU Graz
TU GRAZ Teaching and Learning Service System
All parts linked together over LDAP / Widgets / API / HTTP /
TUGRAZonline
Administration Central Database
TUGTC
Learning Management
ABC-System
CMS
Project
Service
TUGLL
Blogosphere

Figure 5.1: PLE concept. It illustrates the aggregation of different services from distributed university portals and other applications on the WWW.

Section 5.1 gives an overview of the first design structure of the UI and the work flow to understand the functions of the GUI as well as possible. Section 5.2 describes the usability tests that have

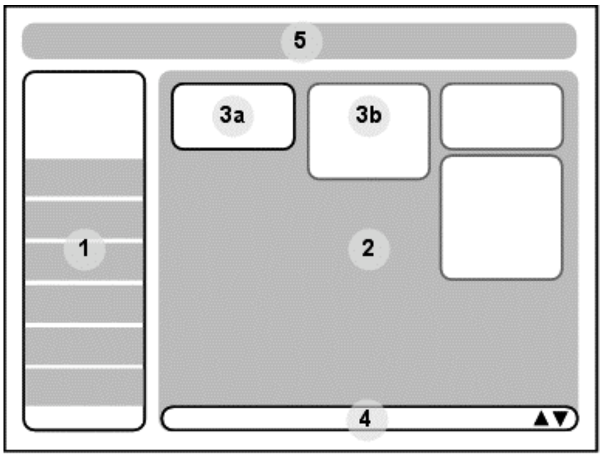


Figure 5.2: PLE User Interface. 1) Sidebar elements contain widget topics. 2) *Widget zone* contains widgets that belong to a widget topic. 3a and 3b) Widgets within the corresponding *widget zone*. 4) Hidden *personal desktop* containing a mash-up of widgets from different widget zones selected by the user. 5) Banner displays information in context of the active widget zone from the network.

been carried out on the described design architecture in the preceding section, at the beginning of the development phase to assure that the end result will be satisfying. Section 5.3 demonstrates the actual prototype of the developed PLE and describes the two main screens. Section 5.4 demonstrates some widget prototypes that have already been implemented in the PLE.

5.1 User Interface Structure

There are many e-Learning services that are already provided by the TU Graz, including course administrations in TUGraz online, course learning materials such as e-books, podcasts, etc., in TUGTC and user-generated contents as well as user contributions such as blogs, bookmarks and file posts in TUGLL. All these services are going to be integrated in the PLE as widgets. Therefore it was necessary to design a coherent GUI to avoid probable usability and consistency problems that may occur [Taraghi et al., 2009b].

The PLE GUI is a combination of a traditional UI with a sidebar element and banner for orientation and navigation. In addition, it offers a widget-based UI with the so-called *"widget zones"*, which require adjustments to be made by the user (see figure 5.2).

The following sections describe each UI element in detail [Taraghi et al., 2009b].

5.1.1 Sidebar

Widgets are categorized according to pre-defined topics. Each widget topic (category) has its own widget zone. The sidebar elements contain the main widget topics and help the user to switch between widget zones. The topics are easily extendible if the number of widgets is increasing. Furthermore, it is planned that the sidebar also updates the user on the status of the widgets by means of color and numerical indicators (see chapter 6.2). The sidebar can be switched off in favour of the unfamiliar widget-based UI and replaced by another navigation element, which resembles the Mac Dock menu on the bottom, left, top or right part of widget zones (see figure 5.4).

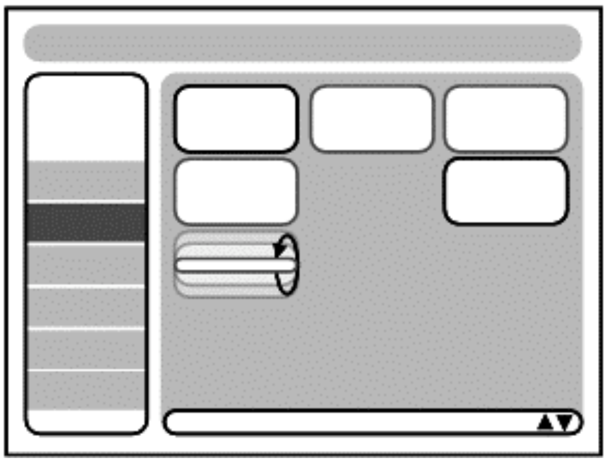


Figure 5.3: PLE User Interface: The widget is flipped to change to its rear side.

5.1.2 Widget Zone

Widget topics include different areas related to formal and informal learning, i.e. *"Communication Center"* for emails, chats and news groups, *"TeachCenter"* for all services related to the TU Graz LMS system TUGTC, such as course materials, podcasts, etc., *"LearnLand"* for services related to the TU Graz blogosphere system TUGLL social bookmarking, file sharing, etc. and *"Help and Support"* for the help desk as well as Frequently Asked Questions (FAQ). These areas are called *widget zones*. Widget zones contain widgets and are structured in columns. Users can switch between widget zones, add, open, close, customize, position and arrange the widgets in different columns according to their personal learning preferences.

5.1.3 Widgets

The widgets consist of a front side and a rear side, where the rear side contains widget preferences that can be modified by the user. If preferences must be changed, the desired widget can be flipped. By this applied flip-animation the users' spatial perception is undisturbed and makes the GUI more understandable (see figure 5.3).

There are two kinds of widgets:

- System widgets: A system widget exists from the very first beginning of the widget zone and its position can be shifted by the user but cannot be removed. Depending on the use case, it contains information relevant to the specific center. An example of system widgets is the *Help* widget. If the widget zone is empty, the system proceeds on the assumption that the user is not familiar with the widget-based UI and needs help. Therefore a *Help* widget is displayed on the widget zone with instructions related to the UI. Once the user adds widgets to the widget zone, the *Help* widget disappears (see figure 5.3 part 3a).

- Standard widgets: Standard widgets are those that can be added by the users from the sidebar and removed later if necessary (see figure 5.3 part 3b).

5.1.4 Personal Desktop

The users are able to create a mash-up of the most frequently used interesting widgets from different widget zones in a special interface called *"personal desktop"*. The *personal desktop* is always available to the user and can be activated at any time. When the user activates the personal desktop, it overlies the

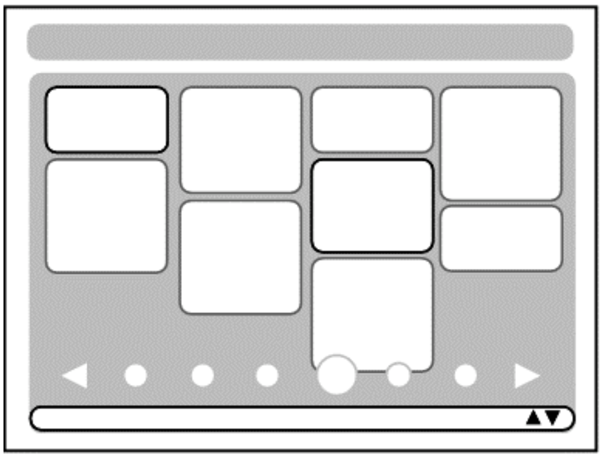


Figure 5.4: PLE User Interface: the *personal desktop* view with sidebar switched off.

whole screen from the bottom of the page upwards (see figure 5.2 part 4). The user can add or remove widgets from all widget zones to his *personal desktop* and arrange them in columns according to his personal taste.

5.1.5 Banner

On top of the page, there is a graphic element called *"banner"* (see figure 5.2 part 5), which contributes to brand a site and helps users to locate contents and orientate themselves. But its main purpose will be to display information from the network in a user-profile-sensitive way (see chapter 6.2). It also keeps track of the currently active widget zone.

5.2 Evaluation of User Interface

From the very beginning, an appropriate and good usability of the PLE was one of the main objectives of the development. Therefore, in the implementation of the first prototype, four tasks are defined to evaluate the functionality of the system and ensure efficient navigation within the PLE. The so-called Heuristic evaluation are carried out to examine opinions of experts in order to reduce problems caused by usability issues to a minimum [Nielsen, 2005].

The first pilot tests are carried out in the concept phase of development to ensure the suitability of test cases. For this reason, a number of experts were asked to answer a basic questionnaire on personal information, PC and Internet experience, and domain knowledge of expert users. The questionnaires were analysed afterwards to ensure a reliable test phase and guarantee a broad range of user types with different levels of domain knowledge and Internet experience. The test environment consisted of a notebook with a built-in webcam, Windows XP Service Pack (SP) 3 as the Operating System (OS) and Camtasia Studio 6.0[61], which was installed for screen capturing. With regard to the different heuristics, a number of different tasks are carried out that are described in tables 5.1, 5.2, 5.3 and 5.4 in more detail. The definition of the tasks followed the crucial steps that a new user of the PLE has to take to get the environment running.

Tables 5.5 and 5.6 show the general outcomes of the heuristic evaluation according to two parameters, namely task completion and task performance, performed by four expert users. According to the test results, the following conclusions could be made:

- Only one user was unable to complete one simple task. This proves that in general, the test users had no problems to carry out the tasks.

Table 5.1: Test case 1 used to evaluate the first PLE prototype

Task 1	**List all courses you are registered for on the TeachCenter (LMS)**
Goal	The expert explores working with the widget-based system. After performing the task, the user should be familiar with the basic navigation process.
Precondition	Expert is logged on. The communication center is open and displays different widgets. All items in the sidebar (hierarchy) are closed. Only the communication center is open.
Expected flow	The user navigates to the sidebar. The user moves the cursor over the TeachCenter area . The user clicks on *TeachCenter*. The widget zone appears The user clicks on *"MyCourses"*
Post Condition	MyCourses-widget is enlarged and all courses are listed.

Table 5.2: Test case 2 used to evaluate the first PLE prototype

Task 2	**Add MyCourses-widget to the personal desktop**
Goal	The user should be able to add a widget to a widget zone or the personal desktop. With this task the user should understand the range of options the personal desktop offers.
Precondition	The user is logged on. MyCourses widget is open.
Expected flow	The user navigates to the top right corner of the widget. The user clicks on "add to Personal Desktop". The personal desktop appears. The personal desktop presents *"Inbox"* widget and *"MyCourses"* widget. The user clicks on the personal desktop bar on the right side "<<" to close.
Post Condition	The personal desktop has 2 widgets (Inbox and MyCourses).

Table 5.3: Test case 3 used to evaluate the first PLE prototype

Task 3	**Try to find the NewsCenter**
Goal	The user should be able to navigate to other centers. The user should experience them from a new viewpoint and discover that each center has its own widget zone.
Precondition	The user is logged on. The user is on the maximized MyCourses widget.
Expected flow	The user moves cursor to the sidebar. The user clicks on NewsCenter. NewsCenter opens its widget zone (RSS).
Post Condition	Widget zone of the NewsCenter is opened.

Table 5.4: Test case 4 used to evaluate the first PLE prototype

Task 4	**Add RSS-widget to NewsCenter widget zone**
Goal	The user should be able to search for available widgets. The user should notice that there are different categories of widgets.
Precondition	The user is logged on. The NewsCenter widget zone appears.
Expected flow	The user notices the ”addWidget” functionality in the widget zone. The user clicks on ”addWidget”. The user receives a list of possible widgets ordered by category (only RSS available). The user clicks on ”addWidget to widget zone”. The list closes. The new RSS-widget is shown in the widget zone. A modal dialogue asks for RSS-URL. The user provides the URL (some example URLs are provided by the experts in the background). The user clicks ”OK”. The RSS-widget loads its content.
Post Condition	The NewsCenter widget zone is extended with another RSS-widget.

Table 5.5: Evaluation results: Task completion

User	**Task 1**	**Task 2**	**Task 3**	**Task 4**
Expert 1	Succeeded	Succeeded	Succeeded	Succeeded
Expert 2	Failed	Succeeded	Succeeded	Succeeded
Expert 3	Succeeded	Succeeded	Succeeded	Succeeded
Expert 4	Succeeded	Succeeded	Succeeded	Succeeded

Table 5.6: Evaluation results: Task performance

User	**Task 1**	**Task 2**	**Task 3**	**Task 4**
Expert 1	00:50	00:10	00:10	00:19
Expert 2	00:19	00:08	00:08	00:50
Expert 3	00:08	00:15	00:20	00:25
Expert 4	00:40	00:20	00:08	00:15
Average	00:44	00:13	00:11	00:12

- The approach to combine a classical navigation bar with the new concept of a widget zone finds general approval. Most users realized the direct relation and consistency of the shown widgets and displayed menu points in the navigation bar.
- The possibility to collect frequently-used widgets on the personal desktop proved to be very popular. The users want to have an overview of things they really need and use most.
- The test results show that widget-based navigation is much more intuitive and provides more overview than the classical hierarchical navigation.
- Some terms caused too many misunderstandings and led to time-consuming problems when trying to accomplish some tasks, e.g. some users referred *personal desktop* to a list of links or favourites.

  Widget was known as a window or module to some users and *RSS* was absolutely unknown to the majority of users with little Internet experience. Some users pointed out that perhaps the term *"Apps"* is more appropriate than *"widgets"* as they were already familiar with the term from mobile devices.
- Many users agreed with the fact that the overview gets lost because widgets appear in different heights. Setting the same initial height for all widgets in the same widget zone would provide a better overview.

According to usability results and to improve the overview, all widgets are configured to have the same initial height when they are closed. Additionally, some metadata about widgets, such as description or name of the corresponding widget developers can be shown to the user in this mode.

5.3 PLE Main Screens

From the end-users point of view, there are only two screens that can be viewed. The start screen, which is publicly accessible on the WWW, and the screens in the logged-in area. These two screens are briefly described below:

5.3.1 Start Page

The start page of PLE[62] is publicly accessible on the WWW under `http://ple.tugraz.at`. It is a static HTML page with a dynamic client-side JavaScript functionality. It contains an embedded video from YouTube[63] that describes the concept and UI of the first prototype in brief. The embedded video was made for the Mediacast contest on Personal Learning Environments in scope of the PLE conference in Barcelona 2010[64] and won the second-best award.

What is more, the start page is also an entry point for authentication. Normally, it shows two login buttons, one for external users and the second for users with a TU Graz access account. Clicking on the first button would show the user the login form. The latter redirects the user to the TU central Identity Provider (IdP) for central authentication. The start page can be adjusted to the user type according to the GET parameter *ref*.

- if *ref=extern*, the login form will be shown directly as it is assumed that an external user wants to sign in.
- if *ref=admin*, the user enters directly to the administration area after successful authentication.
- if *ref=tu*, the user is redirected to the TU central IdP and will not observe the start page.

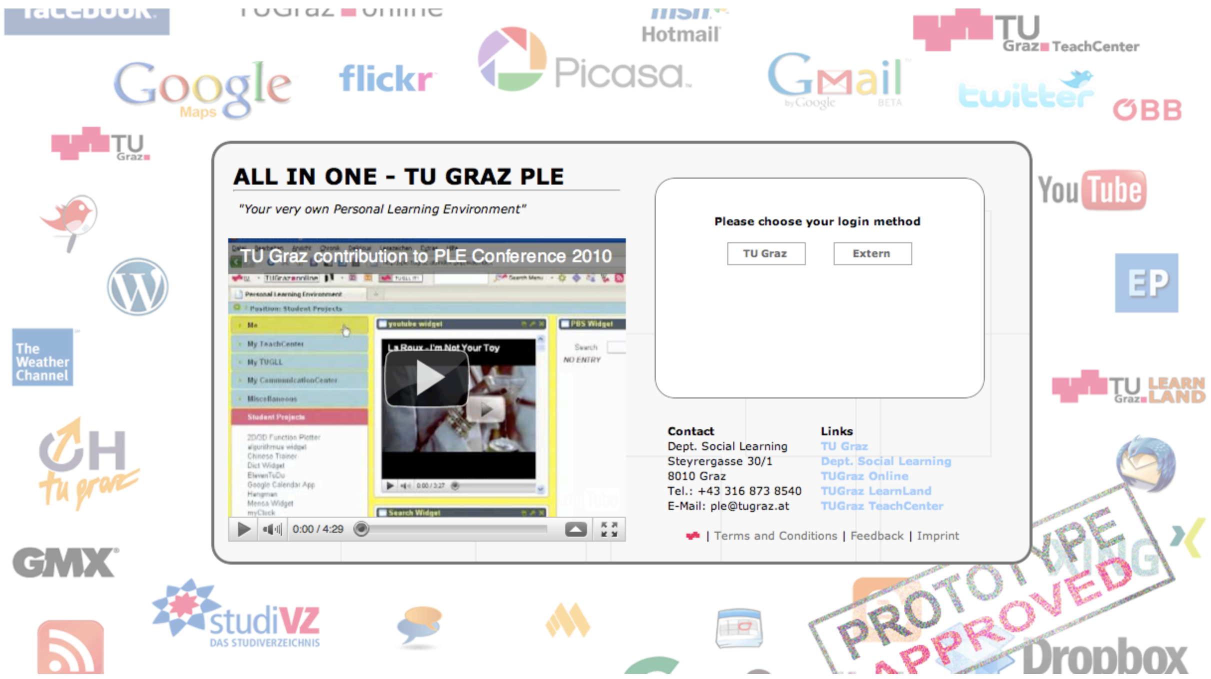


Figure 5.5: PLE start page

If the user is already authenticated in the TU central IdP, he will be authorised as logged in and will be redirected automatically to the widget wall.

Last but not least, the start page controls if the user agent (browser) is supported by PLE. The supported browsers are currently *Chrome* 7+, *FireFox* 3+, *IE* 8, *Opera* 9+ and *Safari* 5+. If the browser is detected as not supported, a corresponding notice appears on the screen to inform the user about eventual malfunctions that may occur.

Figure 5.5 demonstrates the start page of the first prototype.

5.3.2 Logged-in Area

Widget Wall

Once the users are logged in, they will see the main screen of the portal that is called *widget wall*. Since PLE is a RIA and is based on client-server architecture, there is no navigation to any other page. The user would never leave the *widget wall*. From the user point of view, the whole use cases are fulfilled by the client side on *widget wall* (for the detail technical issues for the client side functionality refer to section 3.4).

As described in section 5.1, the *widget wall* screen is divided into three distinct parts: *sidebar* (figure 5.6 part 1), *control panel* (figure 5.6 part 2) and *widgets zones* (figure 5.6 part 3). Figure 5.6 shows the *widget wall* with the *personal desktop* as a widget zone, open.

By clicking on a certain category or personal desktop on the sidebar, a switch is made feasible to the desired widget zone and a list of all widgets within the category is displayed. Users can add widgets to the corresponding widget zone by clicking on a widget label from the widget list. Figure 5.8 shows the sidebar elements. In this example, the widget zone *Miscellaneous* is opened. The widget list within this category is visible.

The sidebar can be dropped out of the screen to enlarge the space for widget zones. As soon as the sidebar is dropped out, the Dock menu appears to allow the user further navigation between widget zones. Figure 5.7 shows this view more obviously.

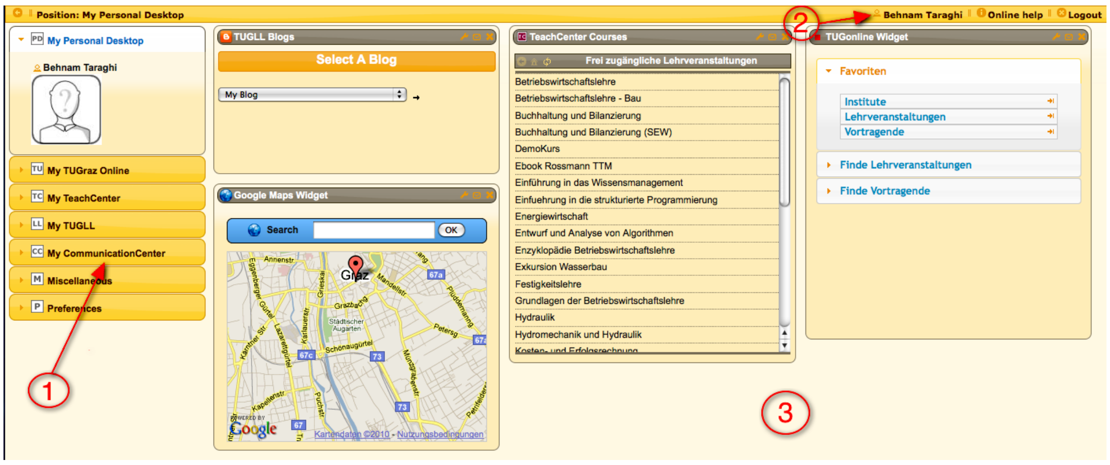


Figure 5.6: PLE widget wall: *sidebar*(1), the *control panel*(2) and the *widgets zone*(3)

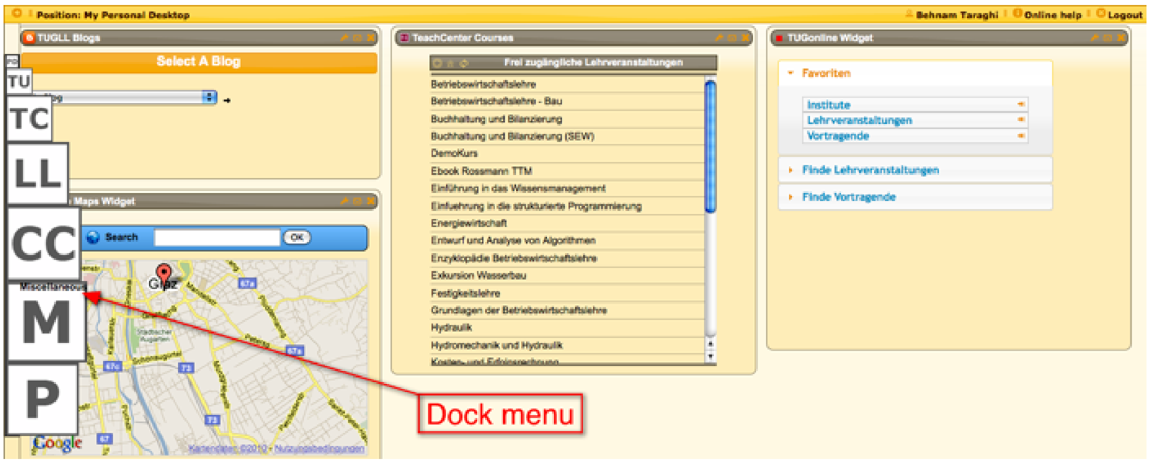


Figure 5.7: PLE dock menu: The widget wall with sidebar dropped out and dock menu opened

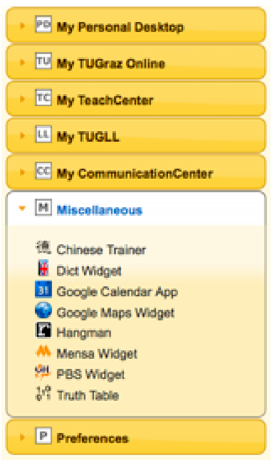


Figure 5.8: PLE Categories: A category in the form of a sidebar element displays the list of widgets

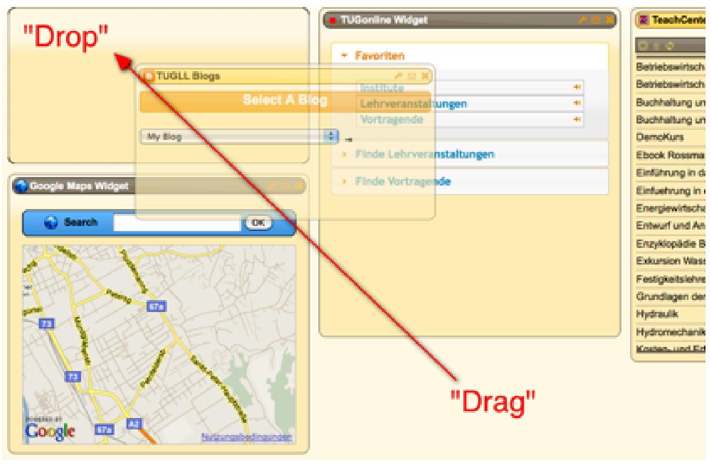


Figure 5.9: PLE widget displacement: Example of moving a widget from one column to another using the "Drag & Drop" technique

For now, the widget zone is organized in 3 columns. It is planned to let the user customize the number of columns, and therefore the ideal view depends on the user's screen resolution (see chapter 6.2). Each column contains an undetermined number of widgets. The columns of widgets are independent of each other and automatically adjust their size depending on the size of the widgets they contain.

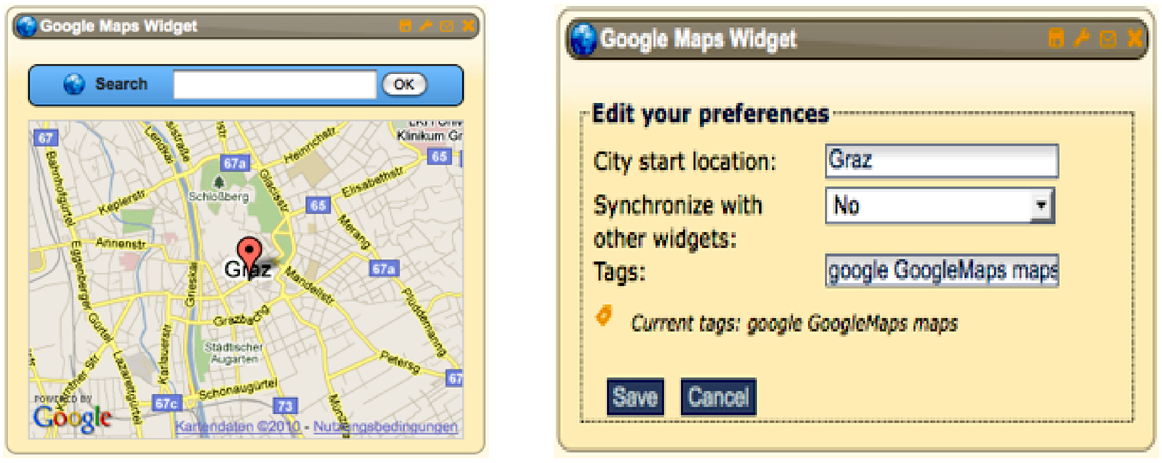
Widgets can be rearranged and displaced from one column to another or within the same column, depending on the user's interests. This is done by *Drag & Drop*. Figure 5.9 shows an example of moving a widget from one column to another by using this technique.

The widget itself contains a toolbar and its main body is loaded within an IFrame. The widget toolbar includes different action buttons that can be used to trigger an action on the widget. These actions can be:

- *Remove a widget*: Removes a widget from the widget zone; user preferences for the corresponding widget will be deleted.
- *Close/Open a widget*: Closes or opens the widget, respectively.
- *Save to personal desktop*: Saves a widget to personal desktop.
- *Edit a widget*: Opens the rear side of the widget with the flipping effect. Users can customize the widget in the edit window. Figure 5.10 shows a Google map widget, adopted from *mywiwall*, with the front side (map) and rear side (edit view).
- *Send an email to the widget developers*: A possibility to get in touch with widget developers.
- *Go to an alternative URL*: Opens the alternative URL that is specified in the widget configuration file, in a new browser window (see section 4.2.2).

Help Page

The help page is a completely static HTML page that provides an introduction on use cases and general functionality of the PLE.



Edit your preferences

City start location: Graz

Synchronize with other widgets: No

Tags: google GoogleMaps maps

Current tags: google GoogleMaps maps

Figure 5.10: PLE Widget *Google Maps*: main and edit window

Administration Pages

If the logged-in user is authenticated as administrator, he is authorised to view the administration pages. They include:

- *User management*: Adding, removing and editing user information.
- *Widget management*: Installing, removing, editing, and modifying widget-related settings.
- *Category management*: Adding, editing and removing categories.
- *Statistics*: Monitoring the actual and archived statistic information (see section 3.2.1).

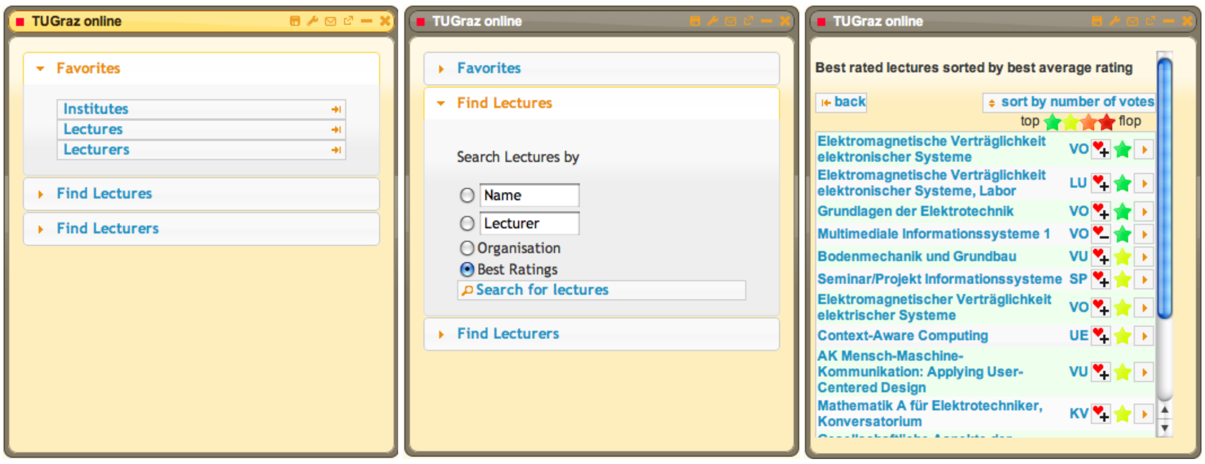
5.4 Widget Prototypes

Except for a couple of widgets, almost all widgets are implemented by using the simple MVC framework described in section 4.5. This approach would help to extend widget functionalities in the future much easier and faster and apply new technologies in widgets, such as HTML 5 features.

Furthermore, the widgets are developed in a way that they allow for changes in style by using a uniform CSS framework. Using a uniform CSS framework in widgets has at least two major benefits: Firstly, the GUI can be developed much faster and easier as the CSS rules are already provided by the applied framework. Secondly, the style sheet of the widget can be switched over to an another one just by using another CSS theme provided by the same framework. As *jQuery UI* is used in PLE for the stylability of PLE-UI, the same framework is also applied in widget development. In the future, switching the style sheet in PLE could affect widgets too as they use the same CSS framework (see section 6.2).

Internationalization is another feature that is supported by the majority of widgets. Although internationalization is already specified in W3C widget packaging and configuration, it is not implemented in the PLE widget engine yet. Hence it is realized by applying a separate solution. Most widgets support English and German as the two main languages that will mostly be required in PLE at TU Graz.

This section will look at the first widget prototypes that have been designed for PLE. Widgets vary from different distributed Internet applications to various services within the university to enhance formal learning and foster informal learning scenarios.



Best rated lectures sorted by best average rating

Elektromagnetische Verträglichkeit elektronischer Systeme	VO
Elektromagnetische Verträglichkeit elektronischer Systeme, Labor	LU
Grundlagen der Elektrotechnik	VO
Multimediale Informationssysteme 1	VO
Bodenmechanik und Grundbau	VU
Seminar/Projekt Informationssysteme	SP
Elektromagnetischer Verträglichkeit elektrischer Systeme	VO
Context-Aware Computing	UE
AK Mensch-Maschine-Kommunikation: Applying User-Centered Design	VU
Mathematik A für Elektrotechniker, Konversatorium	KV

Figure 5.11: PLE Widget: *TUGraz online*

5.4.1 Widgets Representing some University Services

Currently, there are several widgets running in PLE that represent some of the university services. They are described briefly below:

”TUGraz online” widget

”TUGraz online” widget [Bachleitner, 2010] provides some services from the administration system of TU Graz (TUGraz online).

This widget provides users with the possibility to search for courses by name, lecturer’s name, organisation or best ratings. What is more, users can search for lecturers by name or the organisation they belong to. Users can also add the search results to their favourites (institutes, courses and lecturers), so that it won’t be necessary for them to search for this information again. Moreover, users can also rate courses in this widget. The best rated courses are listed on top in the search result.

To describe the back-end briefly, TUGraz online provides a restricted API for third-party applications. The data resources from the administration system are retrieved from TUGraz online API through a cron job in specific time intervals and cached in a local DBMS. The widget communicates with a separate API to retrieve the data needed from local DBMS.

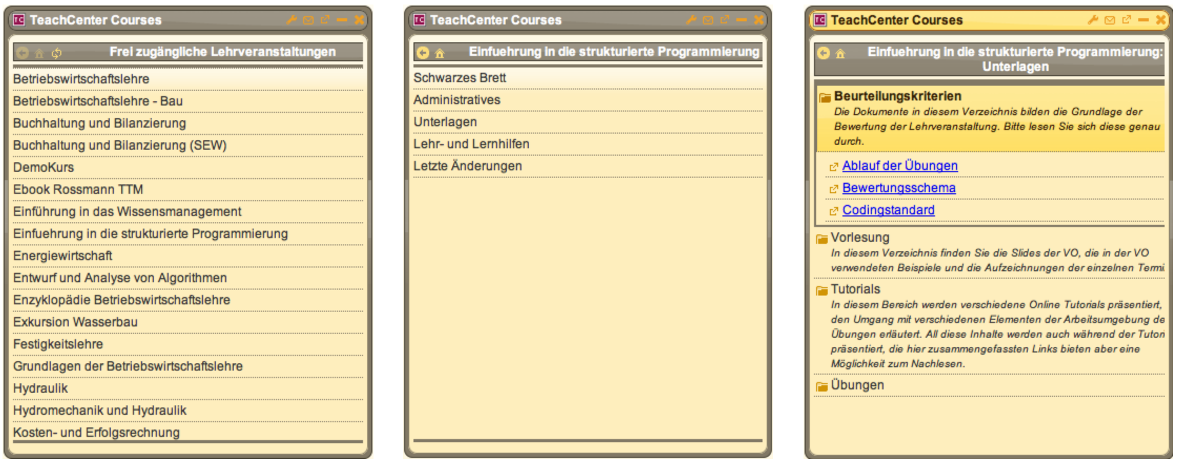
Figure 5.11 shows screenshots of this widget.

”TeachCenter Courses” widget

”TeachCenter Courses” widget represents the LMS of TU Graz, TUGTC.
The user can see a list of existing courses from TUGTC on the start page. The list includes the public courses in TUGTC and the restricted courses which the user is currently registered for. Clicking on a course directs the user to the second page where different options and information regarding the course are listed, such as course materials, announcements, etc. The user can select a certain option to view the required information.

The widget communicates with a separate API. The API retrieves data from TUGTC, parses and converts the results in JSON encoded format and returns it to the client. The API used for this widget can be considered as a server-side application which is responsible for data retrieval from TUGTC.

Figure 5.12 shows three screenshots of this widget.



TeachCenter Courses

Frei zugängliche Lehrveranstaltungen

Betriebswirtschaftslehre
Betriebswirtschaftslehre - Bau
Buchhaltung und Bilanzierung
Buchhaltung und Bilanzierung (SEW)
DemoKurs
Ebook Rossmann TTM
Einführung in das Wissensmanagement
Einfuehrung in die strukturierte Programmierung
Energiewirtschaft
Entwurf und Analyse von Algorithmen
Enzyklopädie Betriebswirtschaftslehre
Exkursion Wasserbau
Festigkeitslehre
Grundlagen der Betriebswirtschaftslehre
Hydraulik
Hydromechanik und Hydraulik
Kosten- und Erfolgsrechnung

TeachCenter Courses

Einfuehrung in die strukturierte Programmierung

Schwarzes Brett
Administratives
Unterlagen
Lehr- und Lernhilfen
Letzte Änderungen

TeachCenter Courses

Einfuehrung in die strukturierte Programmierung: Unterlagen

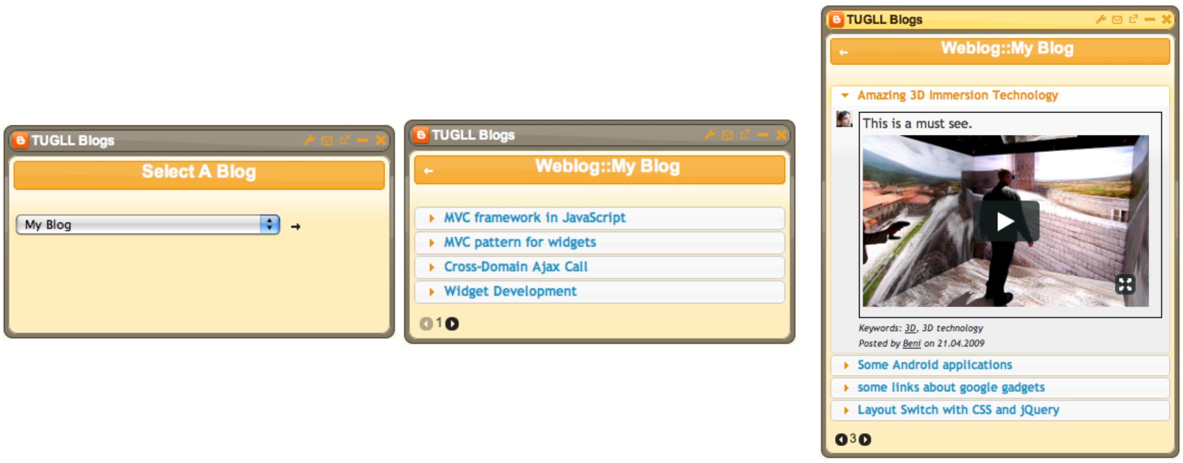
Beurteilungskriterien
Die Dokumente in diesem Verzeichnis bilden die Grundlage der Bewertung der Lehrveranstaltung. Bitte lesen Sie sich diese genau durch.
Ablauf der Übungen
Bewertungsschema
Codingstandard

Vorlesung
In diesem Verzeichnis finden Sie die Slides der VO, die in der VO verwendeten Beispiele und die Aufzeichnungen der einzelnen Termi

Tutorials
In diesem Bereich werden verschiedene Online Tutorials präsentiert, den Umgang mit verschiedenen Elementen der Arbeitsumgebung de Übungen erläutert. All diese Inhalte werden auch während der Tutori präsentiert, die hier zusammengefassten Links bieten aber eine Möglichkeit zum Nachlesen.

Übungen

Figure 5.12: PLE widget *TUGTC* Courses



TUGLL Blogs

Select A Blog

My Blog

TUGLL Blogs

Weblog::My Blog

MVC framework in JavaScript
MVC pattern for widgets
Cross-Domain Ajax Call
Widget Development

1

TUGLL Blogs

Weblog::My Blog

Amazing 3D Immersion Technology

This is a must see.

Keywords: 3D, 3D technology
Posted by Beni on 21.04.2009

Some Android applications
some links about google gadgets
Layout Switch with CSS and jQuery

3

Figure 5.13: PLE Widget: *TUGLL Blogs*

”TUGLL Blogs” widget

”TUGLL Blogs” widget is a blog reader for the blogosphere of TU Graz, TUGLL.

Users can select their own or a community blog on the start screen of the widget. On the second page, users can see a list of selected blogs. They can view the content of each blog item separately.

This widget uses the *RGT* approach for widget authentication and the JSONP API provided by TUGLL for data retrieval. The widget authentication mechanism is described in detail in section 3.3.1. For more information about cross-site requests through JSONP refer to section 4.4.1.

Figure 5.13 shows screenshots of this widget.

”Mail” widget

”Mail” widget acts as an e-mail client for the e-mail account of the university. The university e-mail server is based on the PHP-based Horde Application Framework[65]. In order to realize this widget, a Horde plug-in is developed to handle the JSONP requests of the widget. Figure 5.14 left shows a view of this widget.

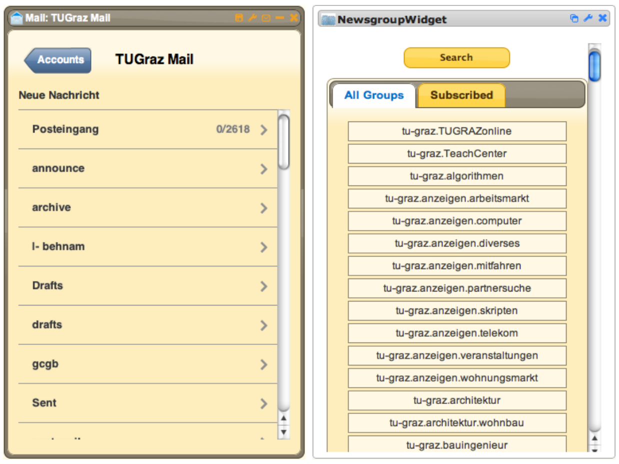


Figure 5.14: PLE Widgets: *Mail* on the left and *Newsgroups* on the right

”Newsgroup” widget

”Newsgroup” widget [Gritsch, 2010] acts as a newsgroup client for newsgroups of the university. Users can read and post threads as well as search and add news groups to their favourites. Figure 5.14 right shows a view of this widget.

5.4.2 Learning Object (LO) Widgets

Learning Object (LO) Widgets are pure client-side widgets (no XHR applied) that can be used for or can help to improve learning, comprehension and understanding of a learning subject. Some examples are mentioned below:

- *TruthTable widget*: A Truth table is a mathematical table used in logics. It is composed of one column for each input variable (for example A and B), and one final column for all the possible results of the logical expression that the truth table is meant to represent (for example, A AND B). Users can operate with the following operators: *NOT*, *AND*, *OR*, *XOR*, => and <=> (see figure 5.15 left).

- *Kana Quiz widget*: The Kana Quiz widget follows the Hepburn romanization (see figure 5.15 middle). Three character data subsets are available: Hiragana, Katakana and Kana (Hiragana + Katakana). Some Kana are not included in the quiz because they are obsolete Kana (e.g. ’wi’ and ’we’). For Hiragana romanization and stroke order, the quiz uses the table specified at `http://en.wikipedia.org/wiki/File:Table_hiragana.svg`, and for Katakana romanization and stroke order the table specified at `http://en.wikipedia.org/wiki/File:Table_katakana.svg`.

- *Chinese Trainer widget*: The Chinese Trainer widget can be used to learn Chinese vocabulary. For each Chinese word you can see the Pinyin and German translation (see figure 5.15 right).

- *Hangman widget*: The Hangman widget represents the Hangman game. The user can select a learn catalogue and try to answer the questions in form of a Hangman game. This widget realizes game-based learning scenarios (see figure 5.16).

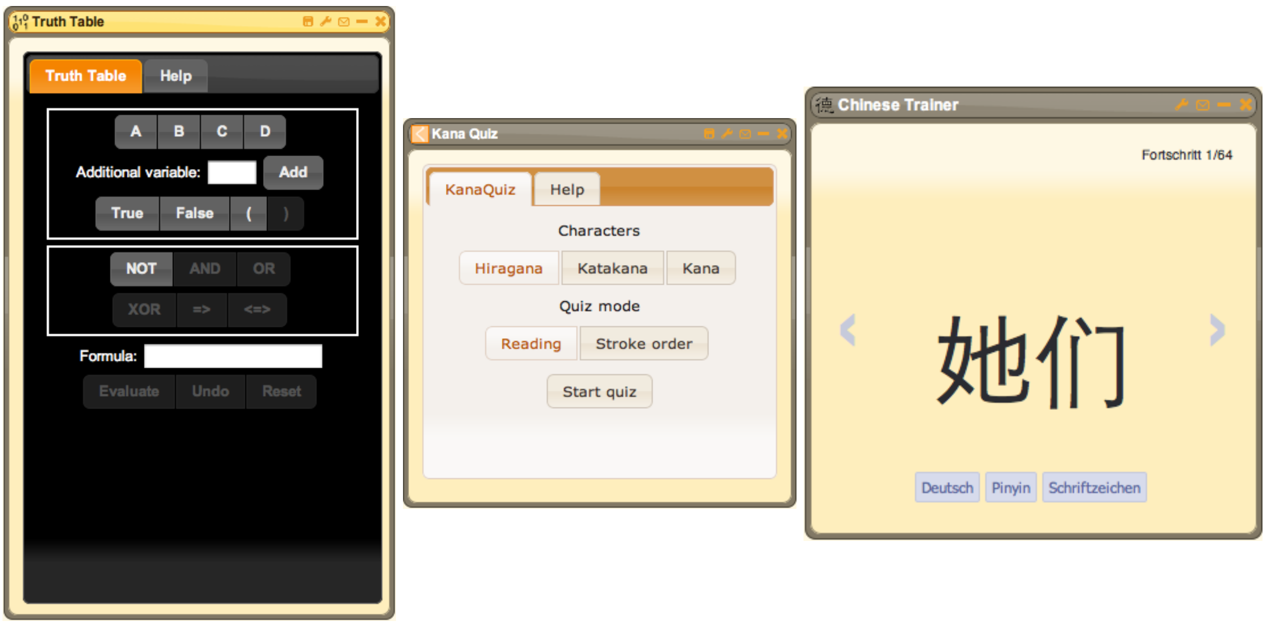
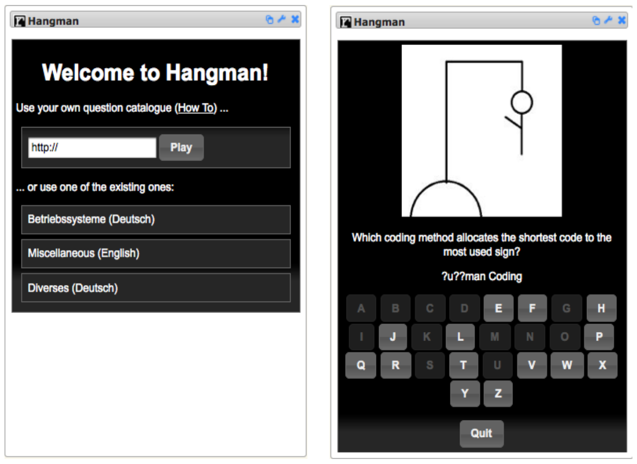


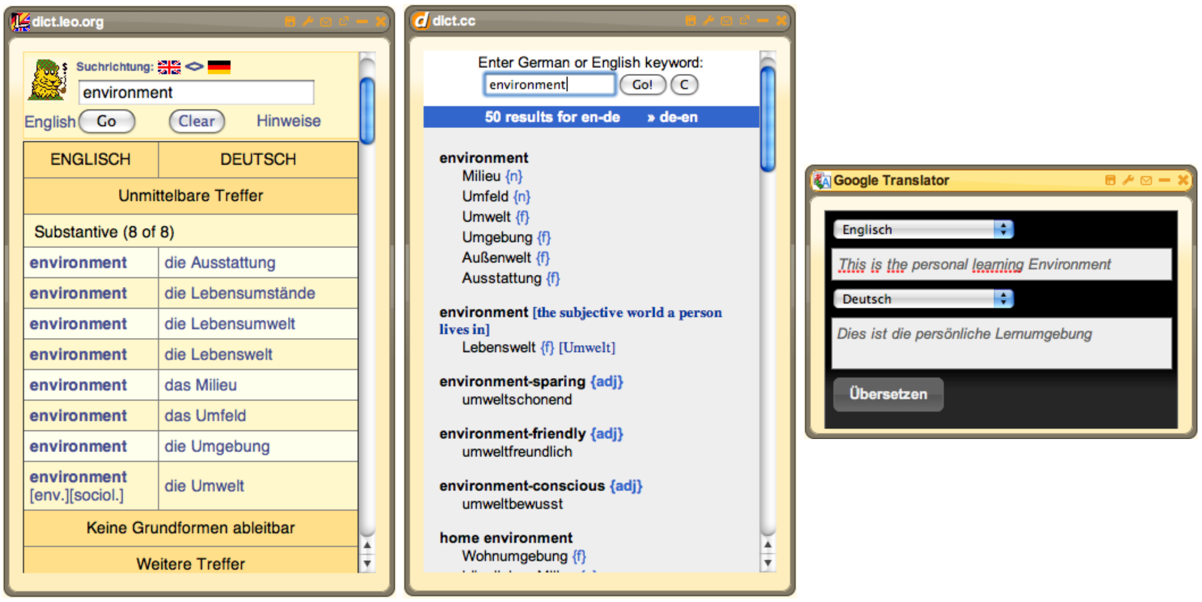
Figure 5.15: PLE LO Widgets: from left to right *TruthTable*, *Kana Quiz* and *Chinese Trainer*



Which coding method allocates the shortest code to the most used sign?

?u??man Coding

Figure 5.16: PLE LO Widget *Hangman*: A game-based learning scenario


dict.leo.org

Suchrichtung:

environment

English Go Clear Hinweise

ENGLISCH	DEUTSCH
Unmittelbare Treffer	
Substantive (8 of 8)	
environment	die Ausstattung
environment	die Lebensumstände
environment	die Lebensumwelt
environment	die Lebenswelt
environment	das Milieu
environment	das Umfeld
environment	die Umgebung
environment [env.][sociol.]	die Umwelt
Keine Grundformen ableitbar	
Weitere Treffer	

dict.cc

Enter German or English keyword:

environment Go! C

50 results for en-de » de-en

environment
Milieu {n}
Umfeld {n}
Umwelt {f}
Umgebung {f}
Außenwelt {f}
Ausstattung {f}

environment [the subjective world a person lives in]
Lebenswelt {f} [Umwelt]

environment-sparing {adj}
umweltschonend

environment-friendly {adj}
umweltfreundlich

environment-conscious {adj}
umweltbewusst

home environment
Wohnumgebung {f}

Google Translator

Englisch

This is the personal learning Environment

Deutsch

Dies ist die persönliche Lernumgebung

Übersetzen

Figure 5.17: PLE Widgets: from left to right *dict.leo.org*, *dict.cc* and *Google Translator*

5.4.3 Widgets Representing Services on the WWW

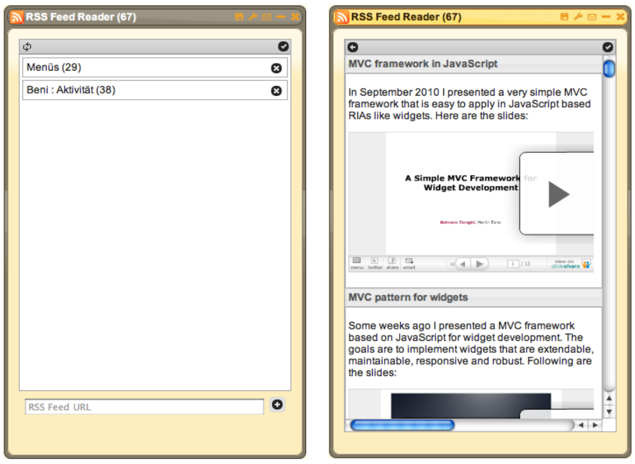
There are many widgets that have been implemented up to now that represent some services on the WWW. Some of the widgets described here are mentioned for the sake of completeness.

- *Dictionary services*: There are already three widgets that apply certain translation services on the WWW. *dict.leo.org widget* uses the online service of `http://dict.leo.org/`. *dict.cc widget* uses the service of `http://www.dict.cc/`. *Google Translator widget* uses the Google translation service to perform translation on `http://translate.google.com/`. With the help of this widget it is possible to translate sentences to numerous languages. Figure 5.17 shows these three widgets.

- *RSSFeedReader*: The RSS Feed Reader widget can be used to read subscribed RSS feeds. The widget shows how many new feeds are still not being read by the user (see Figure 5.18).

- *Social Networks*: Social networks can also be integrated in a PLE. Figure 5.19 shows Twitter [Sandriesser, 2010] and Facebook [Tazl, 2010] widgets as two examples.

  For social network services on the WWW, different widgets can be developed and integrated into a PLE. In case of Twitter for instance, there are lots of third-party applications that provide services, which base on mining twitter data. A widget can provide users with the possibility to search among their own tweets archive by using Grabeeter[66] [Mühlburger et al., 2010] or to perform semantically enriched search queries to gain more accurate information about different entities in twitter data sets [Softic et al., 2010]. In case of Facebook, there are also many possibilities such as chat, messages, pages, etc.



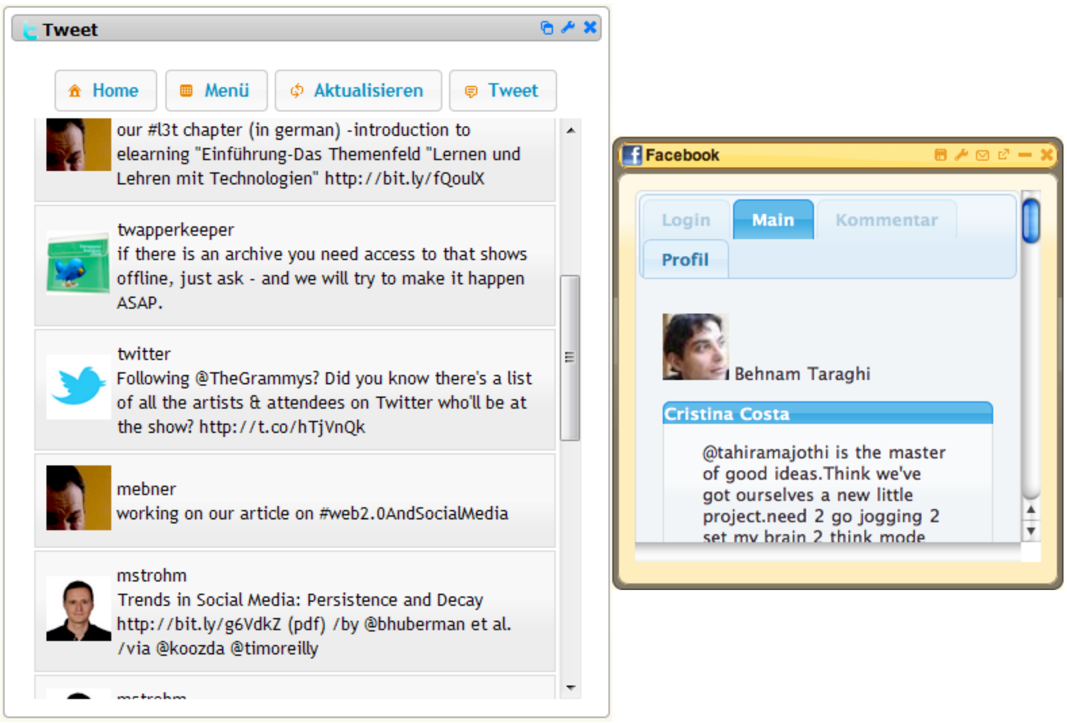
MVC framework in JavaScript

In September 2010 I presented a very simple MVC framework that is easy to apply in JavaScript based RIAs like widgets. Here are the slides:

MVC pattern for widgets

Some weeks ago I presented a MVC framework based on JavaScript for widget development. The goals are to implement widgets that are extendable, maintainable, responsive and robust. Following are the slides:

Figure 5.18: PLE Widget: RSS Feed Reader



our #l3t chapter (in german) -introduction to elearning "Einführung-Das Themenfeld "Lernen und Lehren mit Technologien" http://bit.ly/fQoulX

twapperkeeper
if there is an archive you need access to that shows offline, just ask - and we will try to make it happen ASAP.

twitter
Following @TheGrammys? Did you know there's a list of all the artists & attendees on Twitter who'll be at the show? http://t.co/hTjVnQk

mebner
working on our article on #web2.0AndSocialMedia

mstrohm
Trends in Social Media: Persistence and Decay http://bit.ly/g6VdkZ (pdf) /by @bhuberman et al. /via @koozda @timoreilly

Behnam Taraghi

Cristina Costa

@tahiramajothi is the master of good ideas.Think we've got ourselves a new little project.need 2 go jogging 2 set my brain 2 think mode

Figure 5.19: PLE Widgets Social networks: *Twitter* and *Facebook*

Chapter 6

Outlook

" The distinction between the past, present and future is only a stubbornly persistent illusion."

[Albert Einstein.]

This chapter first looks at some general trends regarding the usage of widgets as tiny applications along with RIAs in general, and further explores some ideas for future work related to the implemented PLE.

6.1 General Trends

With the growth of modern technologies and the reduction of prices for modern devices such as smartphones, the number of users of these devices is increasing steadily. The tendency to use mobile applications is obviously growing rapidly as it is observable at global mobile statistics 2011 [mobiThinking, 2011]. This affects widget-based systems as well. Although mobile widgets have existed for some time now, the lack of uniform standards and support among different mobile browsers for end devices has made an integration of web-based widgets in mobile phones very difficult. However, users tend to access to their tiny applications ubiquitously, which means anywhere and at any time, also when they are mobile. Bringing the PLE to the mobile world would definitely meet the expectations of many users. Especially smart phones, such as iPhone[67] and Android[68] capable mobile phones, should be taken into consideration.

From the technical point of view, the number of so-called web Apps is also increasing. HTML 5 provides many new features that make the realisation of many new interesting use cases in RIAs possible. Web storage[69] (local and session), web SQL database[70] (SQLite[71] engine) within the browser, canvas, audio and video tags, file upload[72] and support for Geolocation[73] as well as Drag and Drop[74] are just some examples of the novelties in this area. It may still take some more time before all browsers will have implemented the new HTML 5 feature mentioned above, but HTML 5 will definitely be the future of web-based RIAs.

6.2 Ideas for Future Work

There are some specific ideas and suggestions for further work, which will be explored in detail in the following sections.

6.2.1 Widget Engine Upgrade

As mentioned in chapter 3, PLE bases upon *mywiwall*, the widget engine implemented within the Palette project. *mywiwall* has implemented W3C widgets packaging and configuration as well as interface specifications, while also adding extensions to support additional features (see section 3.1). Since then, the W3C widget specifications have been updated several times, which means the widget engine used in PLE is not up to date and is not 100% compatible with the actual version of W3C specifications. Therefore an upgrade to the current version would be necessary to support the deployment of other W3C-based widgets in PLE, such as wookie widgets.

6.2.2 Missing W3C Widget Specifications

As mentioned before, the widget engine of PLE implements and extends an old version of W3C widgets packaging and configuration as well as interface specifications. These two main specifications are necessary to adhere to in order to run widgets within an environment based on W3C, like PLE. However, there are some other W3C specifications related to widgets that are described in section 4.1. In order to be fully compatible with W3C, missing specifications must be also implemented.

6.2.3 UI Extensions

PLE stands for Personal Learning Environment, which literally means it should provide as many possibilities for personalization as possible. Moreover, the users should be able to organize their widgets (customize their PLE) according to their own wishes. As a matter of fact, this is already the case to some extent. Users can add, remove, close, open and reorganize widgets within widget walls. However, categories are static entities that are allocated by the administrator. This should be changed so that the users can categorize their widgets according to their own personal opinions and needs.

Furthermore, the number of installed widgets is increasing in the course of time. It may be useful in the future to provide a widget pool in PLE, like the app store for iPhone apps or market for Android apps. Users should be able to search for various widgets in a widget pool. To realize this concept, a detailed semantic description of widgets may be appropriate to apply in order to guarantee that the desired widgets can be found in the widget pool through a semantically enriched search query.

The next issue is the fixed number of columns within widget walls. The number of columns should be customizable as well since users might access their PLE with different screen resolutions. Three columns might be enough for very low resolutions. On the other hand, screens with high resolutions may have enough capacity for more than three columns in widget walls.

The next point is the full-view modus for widgets. Some widgets, i.e. *TUGTC courses* widget, need to display very detailed information, which sometimes requires more space to be shown correctly. It must be possible to enlarge the widget, for instance by a new *"enlarge"* action button on the widget toolbar, so that the user can follow the information or application provided by the widget as easily as possible.

Last but not least, a user-notification system must be integrated into the UI to inform the user about new situations and states in different active widgets. Some examples could be a new incoming email in the mail widget, a new unread RSS feed item in RSS widget, a recently published answer to a user's thread in a newsgroup widget, a new message from a chat widget, etc. Notifications can appear within the sidebar area for each widget wall or on the control panel.

6.2.4 Extension of Simple MVC framework

In section 4.5 a simple MVC framework is introduced that is used in widget development. The framework provides a basis for developers to implement their JavaScript code in a specified structured way, so that it is maintainable and extendible. However, the framework does not provide any tools for dynamic generation of uniform GUI elements. From the usability viewpoint, it may be advantageous to offer widgets in the PLE that use the same UI structure and elements, i.e. paging, sliding, input fields, buttons, etc. In this way, the users will not need to familiarize themselves with the UI structure of each widget individually. To reach this goal, the designed MVC framework must be extended to provide the necessary functionalities regarding the dynamic generation of GUI.

6.2.5 PLE as a Desktop Application

If users work with a web-based application too often, it would be more comfortable and enjoyable if they do not have to open a browser each time they want to access the application. A desktop application would be handier to let the users feel that they have their own PLE on their PC. On the other hand, as mentioned in section 6.1, using HTML 5 web storage would make it possible to realize an offline PLE that can be started directly from the desktop. In this case, once the PLE is loaded, the user can go offline and work with widgets. Nevertheless, a condition that would have to be fulfilled beforehand is that widgets also support a local cache and can run offline.

The easiest way to realize this idea is to use Prism[75]. Prism is a Firefox extension that provides the possibility to run HTML pages as stand-alone applications on the desktop. Mac[76], Linux and Windows[77] OS are supported. Naturally, this approach requires that the Firefox browser is installed on the client machine. According to the last PLE statistics, about 67% of registered users use Firefox to access the PLE. This makes Prism a good candidate to be applied for desktop PLE as majority of users already have Firefox installed on their machines.

Another solution can be JavaFX[78]. JavaFX is a rich client platform for building cross-device applications. JavaFX applications run in a Java Virtual Machine (JVM) and can therefore be executed in a broad range of devices, since JVM is installed on many devices, including PCs and mobile devices. Adobe AIR[79] or Microsoft Silverlight[80] can also be applied [Taraghi et al., 2009b]. They require an own runtime environment being installed on the client though, which renders them rather inappropriate [Taraghi et al., 2009c].

6.2.6 Desktop and Dashboard Widgets

For users with Mac or Windows 7 OS, ”dashboard widgets” or ”desktop widgets” is not an unknown term. According to PLE statistics concerning user agents, Windows 7 OS is leading with a share of ca. 43%, with Mac OS being the runner-up with ca. 23%. It can be deduced from the statistics that about 66% of users have the possibility to use widgets as stand-alone applications on their OS desktop or dashboard, respectively. To support widgets running alone on Windows or Mac OS, a converter is necessary to convert a W3C widget on the fly to the desktop and dashboard widget according to the desktop and dashboard widget specification in Windows and Mac OS, respectively.

6.2.7 Mobile PLE

As mentioned in section 6.1, it would be advantageous to provide a mobile PLE. The mobile version requires a reimplementation of the client-side logic and of course a mobile-suitable UI. The JavaScript functionality needs to be held as light and simple as possible to support a wide range of mobile browsers. For the UI, the *jQuery mobile framework*[81] can be used. It is built upon the *jQuery UI framework* and supports almost all popular mobile device platforms. Although it is currently alpha-released, it will definitely be a good choice for future work.

6.2.8 Other Widget Specifications

In order to have a variety of widgets running in the PLE, it is required that many widget developers are engaged in the development process. If other existing widget specifications, i.e. Google gadgets[82], are supported in the PLE, it is probably easier to attract more developers, as each can have the know-how in one specification. On the other hand, the widgets implemented in other specifications can be applied in the PLE too, which would increase the variety and number of installed widgets in the PLE.

6.2.9 Web Services

The PLE at TU Graz may be provided to other higher educational institutes and universities that follow the same approach as TU Graz in the area of TEL. It would be advantageous to provide web services for communication between different PLEs that are actually all instances of the same one. Different use cases can be realised through web services, i.e. automatic exchange of new widgets, searching for installed widgets on one platform, etc.

6.2.10 Recommender Systems in PLE

A common problem for mashups-based systems, such as PLEs, is the amount of data that is quickly gathered in a short period of time. To overcome the overwhelming effect and help the users to structure and filter the information flow within the PLE, it would be beneficial to investigate some possibilities to apply the recommender system technology within a PLE.

Nowadays, there are many recommender systems that recommend certain web services, applications or widgets to individual users instead of recommending content or persons [Kokash et al., 2007]. Depending on the number of widgets available in the PLE, a recommender widget for the existing widgets can be very useful. Such a widget can suggest certain widgets to particular users. It can also recommend different sets of widgets to the users on the basis of their study domain or course selection. For instance, a student of computer science that subscribed to a course in computer algorithms could take advantage of an algorithm visualization widget. Thus, after subscription to the course, the recommender will suggest this particular widget to the student. It is required that the recommender widget can retrieve the information it needs from other widgets, e.g. the *TUGTC courses* widget must notify the recommender widget about the courses, which the user is already registered for. This can be realised through an inter-widget communication mechanism described in section 3.4.3. The recommender would mainly be based on a top-down knowledge-driven approach, as it is the case in some e-commerce systems [Felfernig, 2005]. It can also be combined with a bottom-up approach, like collaborative filtering, and allow students to assess the value and usability of a widget.

[Drachsler et al., 2010] applied a recommender system approach from e-commerce, implemented in a mashup PLE. Learners can specify different Web 2.0 services in a mashup PLE (the so called ReMashed system) and rate the content to train the recommender system according to their needs and interests. The recommender system suggests the most suitable Web 2.0 items to the learners. Such an approach can also be used in PLE at TUGraz to provide the users with a more personalized usage of web 2.0 related widgets and avoid the overwhelming effect that may arise due to frequently updated information in social and content-sharing Web 2.0 networks such as Twitter, Facebook, YouTube, etc.

Chapter 7

Concluding Remarks

“ Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning. ”

[Winston Churchill.]

The main concern of this thesis was to develop a Personal Learning Environment (PLE) as a mashup of widgets for the TU Graz in the field of Technology-Enhanced Learning (TEL). As described in chapter 3, the widget engine *mywiwall* that is applied in the PLE is developed in the scope of the Palette project. *mywiwall* provides the possibility of customization that is one of the key features of the User Requirements (URs) in PLE. In *mywiwall*, the W3C Widget Configuration specification is extended and some additional default user preference values are added. These values may be modified by users to customize the widgets according to their own needs (see section 4.2). Widgets Interface specifications described in section 4.1.2 are extended too so that a new way of communication between widgets is enabled. Next to HTTP authentication, *mywiwall* also supports a separate widget authentication mechanism in case the widgets are required to be authenticated by third-party services that they make use of or represent (see S3K in section 3.3.1). The inter-widget communication and widget-authentication mechanism can be applied in PLE for widgets representing university services, since a coordination and trust communication between different services is most often a desired use case. On the other hand, the remote widgets supported by *mywiwall* can be applied in the context of a learning environment very usefully. Dynamically generated contents can be integrated and represented in PLE in form of widgets from remote servers.

All these features made *mywiwall* a good choice to be applied in PLE in order to realize the URs and the concept of PLE in general. However, from today’s point of view, the *wookie* widget engine might have been a better choice (see section 3.1). The work on *mywiwall* terminated as the Palette project came to an end. The *wookie* widget engine has been developed further under the Apache Incubator project and has gained a broad community up to now. Other missing W3C widget specifications have been implemented recently in *wookie*. Google Wave API is currently also supported by *wookie*. At the time of PLE development and design phase it was not clear though what will happen with those two European projects in the future.

Personalisation is the key to a more effective learning process. Humans do not only differ in their fingerprints but also in all their personal characteristics, social and learning behaviours. As a result, each learner knows his own individual learning methods, which are the most effective mode of practices for him to achieve the required learning goals. Yet, our traditional education system is based on strict uniform rules and learning/teaching schedules that are adapted to all types of learner groups, as if all learners would have the same learning requirements. The educational plans in education institutes may be appropriate to some extent as humans share a common part of learning behaviours. However, the uncommon parts, which are actually different individual/personal learning requirements, are neglected totally in our education system. As a simple example, learning of a language can be taken into consideration. The mother tongue(s) can be learnt almost perfectly (with a high efficiency), since the children learn individually according to their own personal learning manner, which is unique. Learning a new language as an adult person on the basis of a designed formal education process (such as a language course) would definitely not result in the same efficiency nor lead to the same result. Combining the two approaches would increase the learning results enormously. In case of the example mentioned above, attending a course in a surrounding where the native language is spoken can help the learners enormously to meet their personal learning requirements regarding the new language. Through different situations that occur in daily life among native speakers, learners can learn unknowingly much more efficiently than through the formal educational approach.

Considering other study domains, it is obvious that the personal/individual learning process should be integrated into our formal educational system to improve the efficiency of learning results. PLEs can be of great assistance to fulfil this goal in higher educational institutions, provided that the personal/individual learning needs and requirements of all different user groups are met in PLE. In case of PLE at TU Graz, it means that the provided widgets must quantitatively and qualitatively cover the learning needs of users from different study domains. Otherwise the expected results cannot be achieved.

By the same token, PLEs can be used as Personal Teaching Environments (PTEs) or Personal Working Environments (PWEs), especially in the Information Technology (IT) section. All possible web-based tools that can be applied directly for teaching purposes or indirectly for supporting the teaching process can be integrated into a PTE and customized according to the teacher's interests. Answering the students' questions in newsgroups, keeping contact with students in social networks, course administrations, preparing course materials, etc., are only some examples of possible mashups that can be provided in a PTE. In case the aspect of personalisation of a PLE is not required, it can be reduced to an environment to aggregate different useful applications in the workplace as a PWE. For instance, an IT administrator can aggregate different web-based tools in his PWE to manage his daily work more easily, such as monitoring the performance of different systems, configuration of servers, etc.

Appliance of a PLE in higher education requires a rethinking process and some minor changes in the traditional education concept. Teachers must accept and support the idea of PLEs in general. The students would need to change their learning habits as well. They must be encouraged to detect, search and organise their learning tools according to their individual interests. As mentioned before, it is definitely obligatory for a PLE to be complete and cover different individual interests of users. Many various widgets must be provided for individual user groups (students in different study domains) to make a PLE useful from the user's point of view. PLEs can be compared to smartphones, as smartphones are personal devices used as a mashup of apps. A PLE, which does not provide enough useful widgets for all user groups, is the same as a smartphone that provides few apps to the users. Since the PLE at TU Graz has been launched recently, further widget developments will still take some time.

From the technical point of view, the challenging part of this work was the client-side logic. As the PLE is a RIA, most of the functionality takes place on the client side. Client-side programming with JavaScript is different from server-side programming languages. Asynchronous processes such as XHRs must be handled appropriately and numerous performance issues must be taken into account. Moreover, browser compatibility is another issue that must be considered and tested steadily. The same holds true for widget development.

Finally, according to the points discussed above, it can be concluded that the successful use of a PLE in higher education requires some changes in the students' learning process. In traditional learning activities, teachers provide students with specific resources and learning materials. According to the PLE concept, learners must organize their learning resources themselves and search, find and use the resources they need on their own. Enough various learning resources must be provided within PLE to cover a broad set of users' interests and learning requirements. PLE users must get used to self-organisation as there are no teacher-oriented instructions in this approach. They must be released from formal learning structures, which are fixed by lecturers, and switch to a self-controlled learning process. What is more, there must be no restrictions for providing resources and applications within PLE. Teachers should have the option to recommend some resources from PLE to the students. Nevertheless, if teachers are the only resource providers in a PLE, the PLE becomes an application, as it is common in traditional formal learning environments, where teachers are the sole producers and students the consumers of their resources. As Graham Attwell has pointed out [Attwell, 2007b], a PLE must not be seen as a software application but rather as a new approach to using different technologies for learning in an online self-controlled learning environment.

Appendix A

Appendix

A.1 Structure of PLE RDMS

Table A.1: Structure of Table categories

Field	Type	Null	Default
id	smallint(6)	Yes	NULL
category	varchar(50)	Yes	NULL
position	tinyint(4)	Yes	0

Table A.2: Structure of Table cops

Field	Type	Null	Default
copname	varchar(50)	Yes	NULL
coplabel	varchar(100)	Yes	NULL

Table A.3: Structure of Table interface

Field	Type	Null	Default
userid	mediumint(8)	Yes	NULL
widgetid	varchar(50)	Yes	NULL
column	tinyint(4)	Yes	0
minimized	tinyint(1)	Yes	0
position	smallint(5)	Yes	0

Table A.4: Structure of Table interface_dashboard

Field	Type	Null	Default
userid	mediumint(8)	Yes	NULL
widgetid	varchar(50)	Yes	NULL
column	tinyint(4)	Yes	0
minimized	tinyint(1)	Yes	0
position	smallint(5)	Yes	0

Table A.5: Structure of Table interface_preferences

Field	Type	Null	Default
userid	mediumint(8)	Yes	NULL
widgetid	varchar(50)	Yes	NULL
preference	varchar(50)	Yes	NULL
value	varchar(255)	Yes	NULL

Table A.6: Structure of Table login_data

Field	Type	Null	Default
id	bigint(20)	Yes	NULL
userid	mediumint(8)	Yes	NULL
sessionid	char(32)	Yes	NULL
login	timestamp	Yes	CURRENT_TIMESTAMP
logout	timestamp	Yes	NULL
platform	varchar(20)	Yes	NULL
browser	varchar(20)	Yes	NULL
version	varchar(10)	Yes	NULL
ismobile	tinyint(1)	Yes	NULL

Table A.7: Structure of Table tags

Field	Type	Null	Default
label	varchar(50)	Yes	NULL

Table A.8: Structure of Table tracks

Field	Type	Null	Default
trackid	int(11)	Yes	NULL
userid	mediumint(8)	Yes	NULL
widgetid	varchar(50)	Yes	NULL
timestamp	timestamp	Yes	CURRENT_TIMESTAMP
state	varchar(255)	Yes	NULL

Table A.9: Structure of Table users

Field	Type	Null	Default
id	mediumint(8)	Yes	NULL
username	varchar(50)	Yes	NULL
password	varchar(32)	Yes	NULL
copname	varchar(50)	Yes	NULL
level	tinyint(3)	Yes	0
lang	varchar(32)	Yes	en
openid	longtext	Yes	NULL
style	varchar(50)	Yes	NULL
firstname	varchar(50)	Yes	NULL
lastname	varchar(50)	Yes	NULL
email	varchar(50)	Yes	NULL

Table A.10: Structure of Table widgets

Field	Type	Null	Default
widgetid	varchar(50)	Yes	NULL
widgetname	varchar(50)	Yes	NULL
visible	tinyint(1)	Yes	0
copname	varchar(50)	Yes	NULL
category	smallint(6)	Yes	NULL
description	longtext	Yes	NULL
authkey	tinyblob	Yes	NULL
auto_subscription	tinyint(2)	Yes	NULL

Table A.11: Structure of Table widget_tags

Field	Type	Null	Default
widget	varchar(50)	Yes	NULL
tag	varchar(50)	Yes	NULL
userid	mediumint(8)	Yes	NULL

A.2 Class Diagram of Client Logic

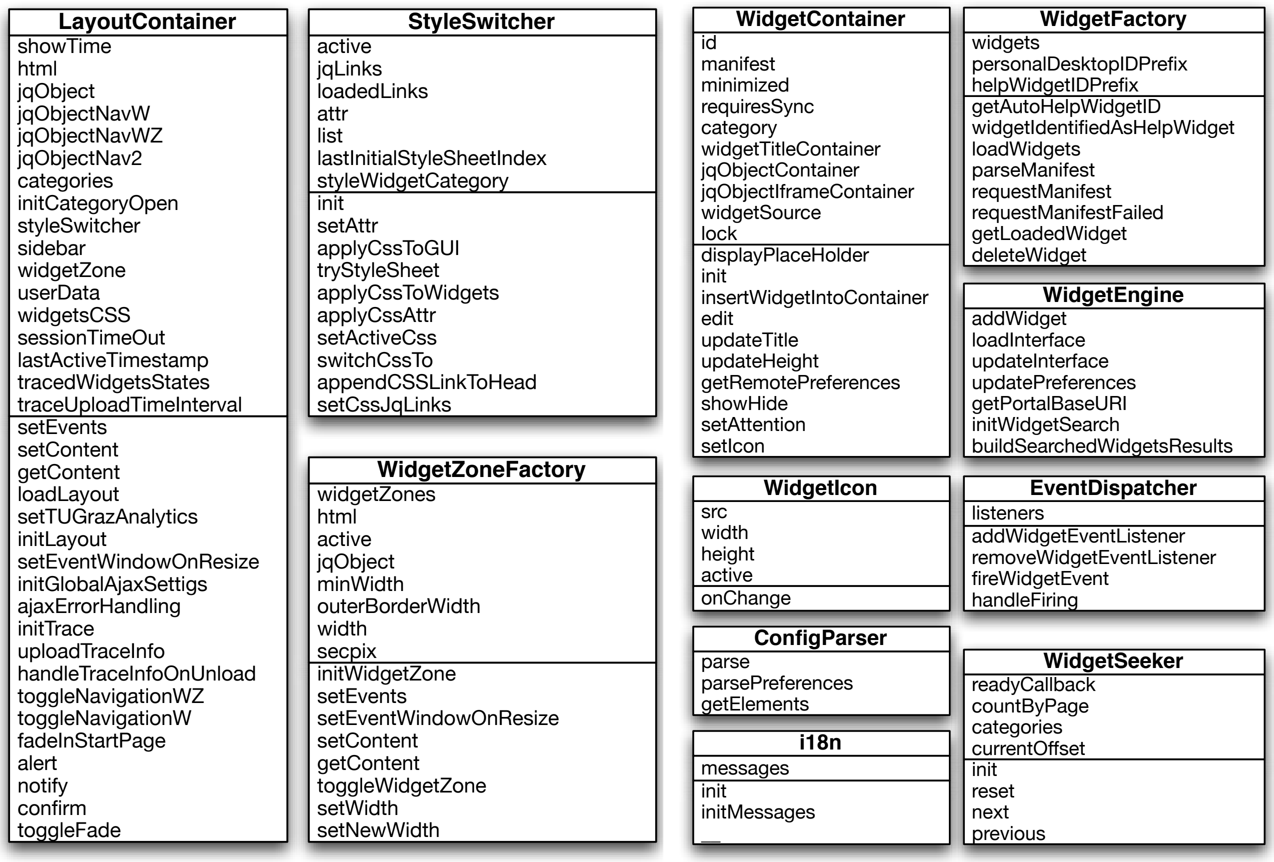


Figure A.1: Class diagram of client logic (1).

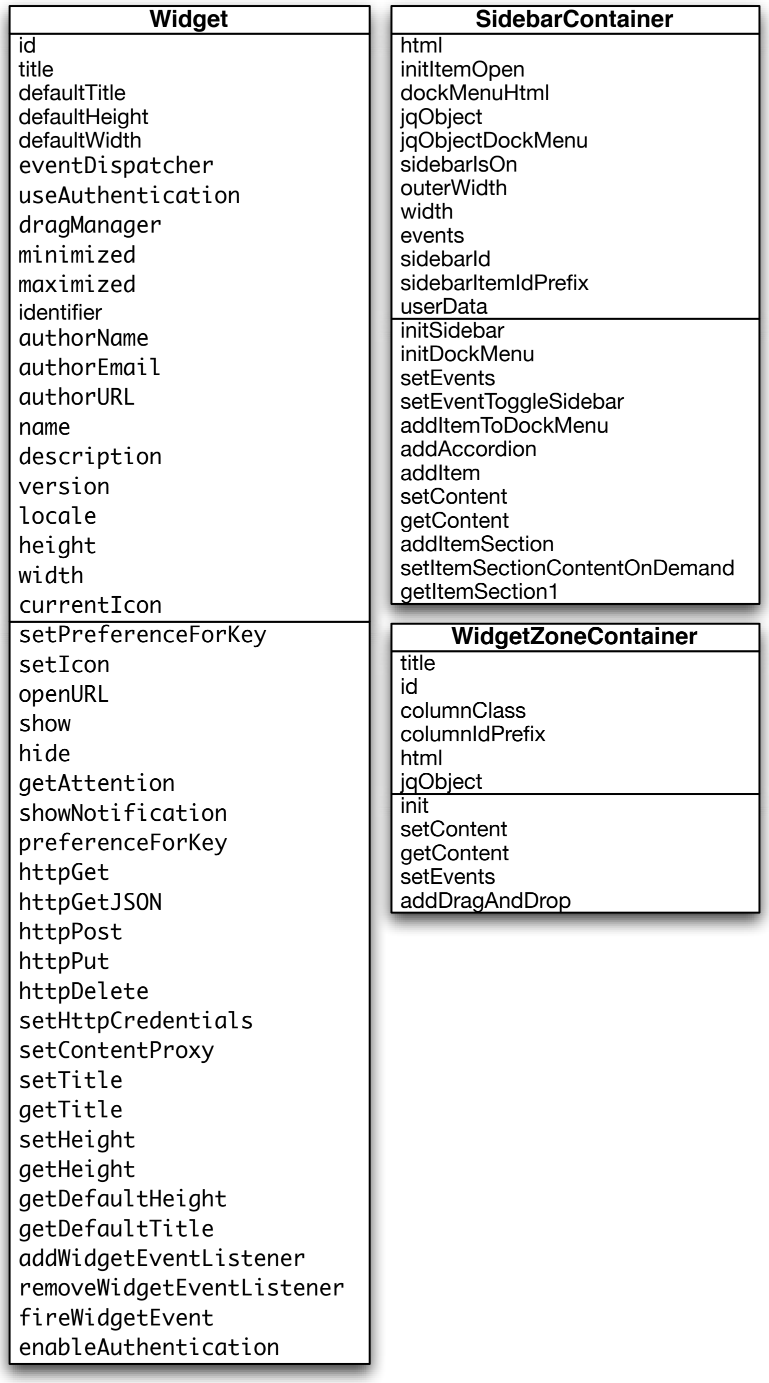


Figure A.2: Class diagram of client logic (2).

A.3 XML Schema Definition (XSD) of Widget Configuration File

A.3.1 manifest.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="http://www.w3.org/TR/widgets/"
  xmlns="http://www.w3.org/TR/widgets/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:palette="http://palette.ercim.org/ns/"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
<xs:import namespace="http://palette.ercim.org/ns/"
  schemaLocation="palette.xsd"/>
<xs:element name="widget">
  <xs:complexType>
    <xs:all>
      <xs:element ref="name" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="title" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="description" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="icon" minOccurs="0"/>
      <xs:element ref="access" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="author" minOccurs="0"/>
      <xs:element ref="license" minOccurs="0"/>
      <xs:element ref="content" minOccurs="0"/>
      <xs:element ref="palette:widget_type" minOccurs="0" />
      <xs:element ref="palette:widget_location" minOccurs="0"/>
      <xs:element ref="palette:alternate_url" minOccurs="0"/>
      <xs:element ref="palette:preferences" minOccurs="0"/>
      <xs:element ref="palette:widget_authentication" minOccurs="0"/>
      <xs:element ref="palette:scrollable" minOccurs="0" maxOccurs="1"/>
    </xs:all>
    <xs:attribute name="id" type="xs:ID" use="required"/>
    <xs:attribute name="version" type="xs:string" use="optional"/>
    <xs:attribute name="height" type="xs:positiveInteger"
      use="optional"/>
    <xs:attribute name="width" type="xs:positiveInteger" use="optional"/>
    <xs:attribute name="start" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="title" type="xs:string"/>
<xs:element name="name" type="xs:string"/>
<xs:element name="description" type="xs:string"/>

<xs:element name="icon">
  <xs:complexType>
      <xs:attribute name="src" type="xs:anyURI"/>
      <xs:attribute name="width" type="xs:positiveInteger"
        use="optional"/>
      <xs:attribute name="height" type="xs:positiveInteger"
        use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="access">
```

```
   <xs:complexType>
       <xs:attribute name="network" type="xs:boolean"/>
       <xs:attribute name="plugins" type="xs:boolean"/>
    </xs:complexType>
 </xs:element>

<xs:element name="author">
   <xs:complexType>
   <xs:simpleContent>
     <xs:extension base="xs:string">
       <xs:attribute name="url" type="xs:anyURI"/>
       <xs:attribute name="email" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
    </xs:complexType>
 </xs:element>

<xs:element name="license">
   <xs:complexType>
   <xs:simpleContent>
     <xs:extension base="xs:string">
       <xs:attribute name="href" type="xs:anyURI"/>
      </xs:extension>
    </xs:simpleContent>
    </xs:complexType>
 </xs:element>

<xs:element name="content">
   <xs:complexType>
     <xs:simpleContent>
       <xs:extension base="xs:string">
         <xs:attribute name="src" type="xs:string"/>
         <xs:attribute name="type" type="xs:string" use="optional"/>
         <xs:attribute name="charset" type="xs:string" use="optional"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
 </xs:element>
 </xs:schema>
```

Listing A.1: XML-schema of the configuration file (manifest.xsd)

A.3.2 palette.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://palette.ercim.org/ns/"
  xmlns="http://palette.ercim.org/ns/">

<xs:element name="scrollable" type="xs:boolean"/>

<xs:element name="widget_type">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="local"/>
      <xs:enumeration value="remote"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="widget_location" type="xs:anyURI"/>
<xs:element name="alternate_url" type="xs:anyURI"/>

<xs:element name="preferences">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="preference"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name="preference">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="enumeration"/>
    </xs:choice>
    <xs:attribute name="name" type="identifier" use="required"/>
    <xs:attribute name="display_name" type="xs:string" use="optional"/>
    <xs:attribute name="datatype" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="string" />
          <xs:enumeration value="bool" />
          <xs:enumeration value="number" />
          <xs:enumeration value="hidden" />
          <xs:enumeration value="enumeration" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="default_value" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="enumeration">
  <xs:complexType>
    <xs:attribute name="value" type="identifier" use="required"/>
    <xs:attribute name="display_value" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>
```

```

<xs:simpleType name="identifier">
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z0-9_]+"/>
   </xs:restriction>
 </xs:simpleType>

<xs:element name="widget_authentication">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="enabled"/>
      <xs:enumeration value="disabled"/>
     </xs:restriction>
   </xs:simpleType>
 </xs:element>
 </xs:schema>
```

Listing A.2: XML-schema of the configuration file (palette.xsd)

Bibliography

Anderson, Terry [2006]. *PLE's versus LMS: Are PLEs ready for Prime time?* http://terrya.edublogs.org/2006/01/09/ples-versus-lms-are-ples-ready-for-prime-time/. (Cited on pages 3 and 4.)

Andrews, Keith [2006]. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Graz University of Technology, Austria. http://ftp.iicm.edu/pub/keith/thesis/. (Cited on page vii.)

Attwell, Graham [2007a]. *E-Portfolios - The DNA of the Personal Learning Environment? Journal of eLearning and Knowledge Society*, 3(2), pages 41–64. http://www.pontydysgu.org/wp-content/uploads/2008/02/eportolioDNAofPLEjournal.pdf. (Cited on page 4.)

Attwell, Graham [2007b]. *The Personal Learning Environments - the future of eLearning? eLearning Papers*, 2(1). ISSN 1887-1542. http://www.elearningeuropa.info/files/media/media11561.pdf. (Cited on pages 4, 6 and 63.)

Augar, N, R Raitman, and W Zhou [2004]. *Teaching and learning online with wikis*. In Atkinson, R, C McBeath, D Jonas-Dwyer, and REditors Phillips (Editors), *Beyond the comfort zone Proceedings of the 21st ASCILITE Conference*, volume 39, pages 95–104. ISSN 00071013. http://www.ascilite.org.au/conferences/perth04/procs/augar.html. (Cited on page 1.)

Aumüller, David and Andreas Thor [2008]. *Mashup-Werkzeuge zur Ad-hoc-Datenintegration im Web. Datenbank-Spektrum*, 8(26), pages 4–10. (Cited on page 6.)

Bachleitner, Stefan [2010]. *Bachelor project at Graz University of Technology*. (Cited on page 51.)

Crockford, Douglas [2008]. *JavaScript: The Good Parts*. O'Reilly Media / Yahoo Press. (Cited on pages 35 and 36.)

Darwin, Charles [1859]. *On the Origin of Species by Means of Natural Selection*, volume 146. John Murray. ISBN 0486450066, 51-52 pages. doi:10.1126/science.146.3640.51-b. http://www.literature.org/authors/darwin-charles/the-origin-of-species/. (Cited on page 21.)

Downes, Stephen [2005]. *E-learning 2.0*. eLearn Magazine. http://www.elearnmag.org/subpage.cfm?section=articles&article=29-1. National Research Council of Canada. (Cited on pages 1 and 3.)

Drachsler, Hendrik, L Rutledge, P Van Rosmalen, H Hummel, D Pecceu, T Arts, E Hutten, and R Koper [2010]. *ReMashed - An Usability Study of a Recommender System for Mash-Ups for Learning*. 5, pages 7–11. ISSN 18630383. doi:10.3991/ijet.v5s1.1191. http://online-journals.org/i-jet/article/view/1191. (Cited on page 60.)

Ebner, Martin [2007]. *E-Learning 2.0 = e-Learning 1.0 + Web 2.0?* In *Proceedings of the 2nd Conference on Availability Reliability and Security ARES07*, pages 1235–1239. IEEE Computer Society. ISBN 0769527752. doi:10.1109/ARES.2007.74. (Cited on page 1.)

Ebner, Martin and Hermann Maurer [2008]. *Can Microblogs and Weblogs change traditional scientific writing?* In *Proceedings of ELearn*, pages 768–776. AACE. `http://go.editlib.org/p/29699`. (Cited on page 1.)

Ebner, Martin, Nicolai Scerbakov, Behnam Taraghi, Walther Nagler, and Isidor Kamrat [2010]. *Teaching and Learning in Higher Education - An Integral Approach.* In Gibson, David and Bernie Dodge (Editors), *Proceedings of Society for Information Technology & Teacher Education International Conference 2010*, pages 428–436. AACE, San Diego, CA, USA. `http://www.editlib.org/p/33375`. (Cited on page 40.)

Ebner, Martin and Behnam Taraghi [2008]. *A Blog Sphere for Higher Education.* In Luca, Joseph and Edgar R. Weippl (Editors), *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2008*, pages 5618–5625. AACE, Vienna, Austria. `http://www.editlib.org/p/29157`. (Cited on page 40.)

Ebner, Martin and Behnam Taraghi [2010]. *Personal Learning Environment for Higher Education - A First Prototype.* In *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2010*, pages 1158–1166. AACE, Toronto, Canada. ISBN 1-880094-81-9. `http://www.editlib.org/p/34779`. (Cited on page 35.)

Ebner, Martin, Behnam Taraghi, and Walther Nagler [2008]. *The TUGLL Plug-ins: Special Needs for a University Wide Blogosphere.* In *Proceedings of International Conference on Knowledge Management and Knowledge Technologies (I-KNOW '08)*, pages 453–456. J.UCS, Graz, Austria. ISSN 0948-6968. `http://i-know.tugraz.at/blog/2008/09/the-tugll-plug-ins-special-needs-for-a-university-wide-blogosphere`. (Cited on page 40.)

Evans, C [2008]. *The effectiveness of m-learning in the form of podcast revision lectures in higher education. Computers & Education*, 50(2), pages 491–498. ISSN 03601315. doi:10.1016/j.compedu.2007.09.016. `http://linkinghub.elsevier.com/retrieve/pii/S0360131507001182`. (Cited on page 1.)

Felfernig, Alexander [2005]. *Koba4MS: Selling complex products and services using knowledgebased recommender technologies.* In *7th IEEE International Conference on E-Commerce Technology*, pages 92–100. Munich, Germany. (Cited on page 60.)

Gamble, M.T. and R. Gamble [2008]. *Monoliths to Mashups: Increasing Opportunistic Assets. Software, IEEE*, 25(6), pages 71–79. ISSN 0740-7459. doi:10.1109/MS.2008.152. (Cited on page 7.)

Gritsch, Hannes [2010]. *Bachelor project at Graz University of Technology.* (Cited on page 53.)

Holzinger, Andreas, Alexander K Nischelwitzer, and Michael D Kickmeier-Rust [2006]. *Pervasive E-Education supports Life Long Learning: Some Examples of X-Media Learning Objects (XLO). Digital Media*, pages 20–26. `http://www.wccee2006.org/papers/445.pdf`. (Cited on page 1.)

Hoyer, Volker [2008]. *Ad-hoc-Software aus der Fachabteilung.* page 98. `http://www.heise.de/artikel-archiv/ix/2008/10/98`. Report - Enterprise Mashups, iX 10. (Cited on page 6.)

Klamma, Ralf, Mohamed Amine Chatti, Erik Duval, Hans Hummel, Ebba Thora, Milos Kravcik, E Law, A Naeve, and P Scott [2007]. *Social Software for Life-long Learning*. *Educational Technology & Society*, 10(3), pages 72–83. ISSN 14364522. https://lirias.kuleuven.be/handle/123456789/164277. (Cited on page 1.)

Kokash, Natallia, Aliaksandr Birukou, and Vincenzo D'Andrea [2007]. *Web Service Discovery Based on Past User Experience*. In Abramowicz, Witold (Editor), *Business Information Systems*, *Lecture Notes in Computer Science*, volume 4439, pages 95–107. Springer Berlin / Heidelberg. doi:10.1007/978-3-540-72035-5_8. (Cited on page 60.)

Krasner, Glenn E and Stephen T Pope [1988]. *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System*. *Journal Of Object Oriented Programming*, 1(3), pages 26–49. http://www.itu.dk/courses/VOP/E2005/VOP2005E/8_mvc_krasner_and_pope.pdf. (Cited on page 34.)

Kulathuramaiyer, Narayanan and Hermann Maurer [2007]. *Current Development of Mashups in Shaping Web Applications*. In *Proceedings of World Conference on Educational Multimedia Hypermedia and Telecommunications 2007*, pages 1172 – 1177. 1, Vancouver, Canada. http://www.editlib.org/p/25525. (Cited on page 2.)

Lubensky, Ron [2006]. *The present and future of Personal Learning Environments (PLE)*. http://www.deliberations.com.au/2006/12/present-and-future-of-personal-learning.html. (Cited on page 3.)

Luca, Joe and Catherine McLoughlin [2005]. *Can blogs promote fair and equitable teamwork?* In *ascilite 2005 Balance Fidelity Mobility maintaining the momentum*. http://www.ascilite.org.au/conferences/brisbane05/blogs/proceedings/45_Luca.pdf. (Cited on page 1.)

Lucky, Robert W [2009]. *To Twitter Or Not to Twitter?* *IEEE Spectrum*, 46(1), page 22. ISSN 00189235. http://search.ebscohost.com/login.aspx?direct=true&db=buh&AN=36042606&site=ehost-live. (Cited on page 1.)

Mason, Robin and Frank Rennie [2007]. *Using Web 2.0 for learning in the community*. *The Internet and Higher Education*, 10(3), pages 196–203. ISSN 10967516. doi:10.1016/j.iheduc.2007.06.003. http://linkinghub.elsevier.com/retrieve/pii/S1096751607000383. (Cited on page 1.)

mobiThinking [2011]. *Global mobile statistics 2011 : all quality mobile marketing research, mobile Web stats, subscribers, ad revenue, usage, trends*. http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats. (Cited on page 57.)

Mühlburger, Herbert, Martin Ebner, and Behnam Taraghi [2010]. *@twitter Try out #Grabeeter to Export, Archive and Search Your Tweets*. In *Proceedings of the 2nd International Workshop on Research 2.0. At the 5th European Conference on Technology Enhanced Learning (ECTEL'10): Sustaining TEL*, volume 675, pages 76–85. CEUR-WS, Barcelona, Spain. ISSN 1613-0073. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-675/. (Cited on page 55.)

Nagler, Walther and Martin Ebner [2009]. *Is Your University Ready For the Ne(x)t-Generation?* In Siemens, George and Catherine Fulford (Editors), *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2009*, pages 4344–4351. AACE, Honolulu, HI, USA. ISBN 1-880094-73-8. http://www.editlib.org/p/32114. (Cited on page 2.)

Naudet, Yannick, Nikos Karousos, Stéphane Sire, Jérôme Bogaërts, Jean-David Labails, Alain Vagner, Marie-Laure Watrinet, Géraldine Vidou, Sami Miniaoui, Manolis Tzagarakis, George Gkotsis, and

Nikos Karacapilidis [2008]. *Final version of PALETTE registry and delivery framework.* http://palette.ercim.org/images/stories/DocumentPDF/d.imp.07_final.pdf. (Cited on pages 12, 13, 20, 27, 29, 30, 31 and 32.)

Nielsen, Jakob [2005]. *How to Conduct a Heuristic Evaluation.* http://www.useit.com/papers/heuristic/heuristic_evaluation.html. (Cited on page 43.)

Olivier, Bill and Oleg Liber [2001]. *Lifelong Learning: The Need for Portable Personal Learning Environments and Supporting Interoperability Standards.* 20. http://ssgrr2002w.atspace.com/papers/14.pdf. (Cited on page 3.)

O'Reilly, Tim [2005]. *What is Web 2.0 - Design Pattern and Business Models for the next Generation of Software.* http://oreilly.com/web2/archive/what-is-web-20.html. (Cited on page 1.)

PALETTE [2008]. *Pedagogically sustained Adaptive Learning through the Exploitation of Tacit and Explicit Knowledge.* http://palette.ercim.org/. (Cited on pages vii, 2, 7, 8 and 24.)

Sandriesser, Jörg [2010]. *Bachelor project at Graz University of Technology.* (Cited on page 55.)

Schaffert, Sandra and Wolf Hilzensauer [2008]. *On the way towards Personal Learning Environments : Seven crucial aspects. eLearning Papers*, 9, pages 1–11. ISSN 18871542. http://www.elearningeuropa.info/files/media/media15971.pdf. (Cited on page 4.)

Schaffert, Sandra and Marco Kalz [2009]. *Persönliche Lernumgebungen: Grundlagen, Möglichkeiten und Herausforderungen eines neuen Konzepts.* In Wilbers, Karl and Andreas Hohenstein (Editors), *Handbuch ELearning*, volume Gruppe 5, pages 1–24. Deutscher Wirtschaftsdienst (Wolters Kluwer Deutschland). http://dspace.learningnetworks.org/handle/1820/1573. (Cited on pages 2 and 4.)

Softic, Selver, Martin Ebner, Herbert Mühlburger, Thomas Altmann, and Behnam Taraghi [2010]. *@twitter Mining #Microblogs Using #Semantic Technologies.* In *Proceedings of 6th Workshop on Semantic Web Applications and Perspectives (SWAP 2010)*, pages 1–12. Bressanone, Italy. (Cited on page 55.)

Softic, Selver, Behnam Taraghi, and Wolfgang Halb [2009]. *Weaving Social E-Learning Platforms Into the Web of Linked Data.* In *Proceedings of International Conference on Semantic Systems (I-SEMANTICS '09)*, pages 559–567. J.UCS, Graz, Austria. (Cited on page 40.)

Taraghi, Behnam and Martin Ebner [2010]. *A Simple MVC Framework for Widget Development.* In *Proceedings of the 3rd Workshop on Mashup Personal Learning Environments (MUPPLE10). In conjunction with the 5th European Conference on Technology-Enhanced Learning (ECTEL'10): Sustaining TEL*, volume 638, pages 1–8. CEUR-WS, Barcelona, Spain. ISSN 1613-0073. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-638/. (Cited on page 34.)

Taraghi, Behnam, Martin Ebner, and Sandra Schaffert [2009a]. *Personal Learning Environments for Higher Education: A Mashup Based Widget Concept.* In *Proceedings of the Second Workshop on Mashup Personal Learning Environments (MUPPLE09). In conjunction with the 4th European Conference on Technology-Enhanced Learning (ECTEL'09): Sustaining TEL*, volume 506, pages 1–8. CEUR-WS, Nice, France. ISSN 1613-0073. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-506/. (Cited on pages 6 and 7.)

Taraghi, Behnam, Martin Ebner, Gerald Till, and Herbert Mühlburger [2009b]. *Personal Learning Environment - A Conceptual Study.* In *Proceedings of International Conference on Interactive Computer Aided Learning (ICL)*, pages 1–10. Auer, M., Villach, Austria. ISBN 978-3-89958-481-3. (Cited on pages 6, 7, 8, 41 and 59.)

Taraghi, Behnam, Herbert Mühlburger, Martin Ebner, and Walther Nagler [2009c]. *Will Personal Learning Environments Become Ubiquitous through the Use of Widgets?* In *Proceedings of International Conference on Knowledge Management and Knowledge Technologies (I-KNOW '09)*, pages 329–335. J.UCS, Graz, Austria. `http://i-know.tugraz.at/blog/2009/09/will-personal-learning-environments-become-ubiquitous-through-the-use-of-widgets`. (Cited on pages 8, 24 and 59.)

Tazl, Oliver [2010]. *Bachelor project at Graz University of Technology*. (Cited on page 55.)

Tuchinda, Rattapoom, Pedro Szekely, and Craig A. Knoblock [2008]. *Building Mashups by example*. In *Proceedings of the 13th international conference on Intelligent user interfaces (IUI 08)*, pages 139–148. ACM, Gran Canaria, Spain. ISBN 978-1-59593-987-6. doi:10.1145/1378773.1378792. `http://www.isi.edu/integration/papers/tuchinda08-iui.pdf`. (Cited on page 2.)

Van Harmelen, Mark [2006]. *Personal Learning Environments*. *Sixth IEEE International Conference on Advanced Learning Technologies ICALT06*, 16(1), pages 815–816. ISSN 10494820. doi:10.1109/ICALT.2006.1652565. `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1652565`. (Cited on page 3.)

Van Harmelen, Mark [2008]. *Design trajectories: four experiments in PLE implementation*. *Interactive Learning Environments*, 16(1), pages 35–46. doi:10.1080/10494820701772686. (Cited on page 3.)

WidgetSpecs [2008]. *W3C Widgets Family of Specifications*. `http://www.w3.org/2008/webapps/wiki/WidgetSpecs`. (Cited on page 25.)

Wild, Fridolin, Felix Mödritscher, and Steinn E. Sigurdarson [2008]. *Designing for change: Mash-up Personal Learning Environments*. *eLearning Papers*, 9, pages 1–15. `http://www.elearningeuropa.info/files/media/media15972.pdf`. (Cited on page 2.)

Wilson, Scott [2005]. *Architecture of Virtual Spaces and the Future of VLEs*. PowerPoint slides. `http://www.cetis.ac.uk/members/scott/resources/itslearning.ppt`. (Cited on page 3.)

Acronyms

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CMS	Content Management System
CORS	Cross-Origin Resource Sharing
CRP-HT	Centre de Recherche Public Henri Tudor
CSS	Cascading Style Sheets
DBMS	Database Management System
DHTML	Dynamic HTML
DOM	Document Object Model
EU	European Union
FAQ	Frequently Asked Questions
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HTTP Secure
ID	Identifier
IdP	Identity Provider
IE	Internet Explorer
IFrame	Inline Frame
IICM	Institute for Information Systems and Computer Media
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
JSONP	JSON with Padding
JST	JavaScript Templates

JVM	Java Virtual Machine
LMS	Learning Management System
LO	Learning Object
MVC	Model View Controller
OO	Object-Oriented
OS	Operating System
PC	Personal Computer
PHP	Hypertext Preprocessor
PLE	Personal Learning Environment
PTE	Personal Teaching Environment
PWE	Personal Working Environment
RGT	Randomly Generated Token
RDMS	Relational Database Management System
RIA	Rich Internet Application
RSS	Really Simple Syndication
SOA	Service Oriented Architecture
SOP	Same-Origin Policy
SP	Service Pack
SQL	Structured Query Language
SSO	Single Sign-on
S3K	Secret Shared Service Key
TEL	Technology-Enhanced Learning
TU	Technical University
TUGLL	TU Graz LearnLand
TUGTC	TU Graz TeachCenter
UI	User Interface
UR	User Requirement
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VLE	Virtual Learning Environment
WDE	Widget Development Environment

WWW	World Wide Web
W3C	WWW Consortium
XHR	XMLHttpRequest
XHTML	eXtensible HTML
XML	eXtensible Markup Language
XSD	XML Schema Definition
YQL	Yahoo! Query Language
ZID	Zentraler Informatikdienst

Further References (last visited: February 2011)

[1]http://twitter.com/

[2]http://www.facebook.com/

[3]http://www.youtube.com/

[4]http://www.slideshare.net/

[5]http://www.scribd.com/

[6]http://www.delicious.com/

[7]http://getwookie.org/Welcome.html

[8]http://wordpress.org/

[9]http://moodle.org/

[10]http://elgg.org/

[11]http://shibboleth.internet2.edu/

[12]http://code.google.com/p/clearfw/

[13]http://www.tudor.lu/

[14]http://clearbricks.org/

[15]http://www.mysql.com/

[16]http://www.postgresql.org/

[17]http://www.sqlite.org/

[18]http://jquery.com/

[19]http://jqueryui.com/docs/Theming/API

[20]http://jqueryui.com/

[21]http://jqueryui.com/themeroller/

[22]http://httpd.apache.org/

[23]http://www.google.com/ig

[24]http://www.netvibes.com/

[25]http://www.protopage.com/

[26]http://www.pageflakes.com/

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

[28]http://www.w3.org/TR/widgets-apis/

[29]http://www.w3.org/TR/widgets-digsig/

[30]http://www.w3.org/TR/widgets-updates/

[31]http://www.w3.org/TR/widgets-access/

[32]http://www.w3.org/TR/widgets-uri/

[33]http://www.w3.org/TR/view-mode/

[34]http://www.w3.org/TR/widgets-reqs/

[35]http://www.w3.org/TR/widgets-land/

[36]http://www.pkware.com/documents/casestudies/APPNOTE.TXT

[37]http://developer.yahoo.com/yql/

[38]http://nb.io/hacks/csshttprequest

[39]http://en.wikipedia.org/wiki/Data:_URI_scheme

[40]http://code.google.com/p/crossxhr/wiki/CrossXhr

[41]http://flxhr.flensed.com/

[42]http://www.w3.org/2008/webapps/

[43]http://www.w3.org/TR/cors/

[44]http://msdn.microsoft.com/en-us/library/cc288060%28VS.85%29.aspx

[45]https://developer.mozilla.org/En/HTTP_access_control

[46]http://javascriptmvc.com/

[47]http://code.google.com/p/trimpath/wiki/TrimJunction

[48]http://rubyonrails.org/

[49]http://trac.puremvc.org/PureMVC_JS/

[50]http://www.sproutcore.com/

[51]http://code.google.com/p/trimpath/wiki/JavaScriptTemplates

[52]http://beebole.com/pure/

[53]http://code.google.com/intl/de-DE/closure/templates/

[54]http://github.com/nje/jquery-tmpl

[55]http://www.jstorage.info/

[56]http://www.prototypejs.org/

[57]http://mootools.net/

[58]https://online.tugraz.at/

[59]http://tugtc.tugraz.at/

[60]http://tugll.tugraz.at/

[61]http://www.techsmith.com/camtasia/

[62]http://ple.tugraz.at

[63]http://www.youtube.com/

[64]http://pleconference.citilab.eu/

[65]http://www.horde.org/

[66]http://grabeeter.tugraz.at/

[67]http://www.apple.com/iphone/

[68]http://www.android.com/

[69]http://www.w3.org/TR/webstorage/

[70]http://www.w3.org/TR/webdatabase/

[71]http://www.sqlite.org/

[72]http://www.w3.org/TR/FileAPI/

[73]http://www.w3.org/TR/geolocation-API/

[74]http://dev.w3.org/html5/spec/dnd.html

[75]http://prism.mozilla.com/

[76]http://www.apple.com/mac/

[77]http://www.microsoft.com/WINDOWS/

[78]http://www.sun.com/software/javafx/

[79]http://www.adobe.com/de/products/air/

[80]http://www.silverlight.net/

[81]http://jquerymobile.com/

[82]http://code.google.com/intl/de-DE/apis/gadgets/