Collaborative Development of a PLE for Language Learning

D. Renzel¹, C. Höbelt⁰, D. Dahrendorf⁵, M. Friedrich³, F. Mödritscher⁴, K. Verbert⁵, S. Govaerts⁵, M. Palmér⁶ and E. Bogdanov⁷

¹ RWTH Aachen University, Aachen, Germany
² ime information multimedia communication AG, Saarbrücken
³ Fraunhofer Institute for Applied Information Technology, St. Augustin
⁴ Vienna University of Economics and Business, Vienna, Austria
⁵ Katholieke Universiteit Leuven, Leuven, Belgium
⁶ Uppsala University, Uppsala, Sweden
⁷ École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

Abstract—This paper provides a report on the experimental collaborative and distributed development of a prototype Widget-based PLE. The development process is described and detailed taking into account the requirements of a language learning scenario. First results are presented, and developer experiences are discussed critically with a focus on the development process as well as problems with current Widget technologies and interoperability.


I. INTRODUCTION

Current endeavors in the domain of Technology Enhanced Learning (TEL) exhibit the need for increased openness and responsiveness of current learning environments. While older learning technology generations were often central and closed systems, often merely focusing on the management of learning processes, the next generation of Personal Learning Environments (PLEs) tackled by the ROLE project ¹ concentrate rates on a highly distributed approach drawing on the combination of established open-standard Web technologies in order to enable the learner-side integration of services and tools from a plethora of heterogeneous sources into customized learning environments. One of the major goals of ROLE is to deliver an appropriate technical infrastructure for the establishment of such responsive open learning environments. Another goal of the project is to establish a community of open source and external developers, contributing further tools and services based on this infrastructure. During the first project developer meeting, we thus agreed to work in parallel to the standard project plan towards a common goal, called the "Christmas Project" with the following objectives in mind:

• Create a first demonstrator of ROLE to visualize the potential of the project
• Enable the consortium to better define needs and derive technical specifications

While the first two objectives address the consortium’s internal collaboration, the last two clearly address a broader audience. Thus, this paper reports on the results of the Christmas Project with a focus on the collaborative distributed development process and a first integrated PLE prototype resulting from this process. It gives the reader an insight into the challenges we faced during our work regarding the development process and the technologies we experimented with. After drawing the conclusion that current technologies and development processes are often still insufficient for a seamless independent development of interoperating learning services and tools, it outlines possible improvements.

The rest of this document is structured as follows. In Section II we describe the development process to give an insight into how our work was organized. In Section III we present details on the requirements elicited for an integrated PLE based on a language learning scenario. In Section IV we present the individual partner contributions in more detail. Section V presents our experiences and a critical conclusion of our work regarding current issues regarding Widget technology. In Section VI we end with a short summary and give a short outlook to further work.

II. DEVELOPMENT PROCESS

Following the objective of establishing a common universe-oriented development process, we planned to explore such a process in a smaller scale within the consortium starting off with a community of nine partners across Europe, from both industry and academia with different degrees of technical background. Since heavy-weight processes would in practice not be feasible and accepted with a large-scale developer community, we decided to keep the process as lightweight as possible, having fewer rowing concepts from standard processes such as Agile Development [1][2], e.g. short iterations cycles, shared code & documentation, continuous integration, regular developer meetings.

References

communication, etc. However, given by the spatial distribution of the community, concepts requiring physical attendance emphasized in Agile approaches had to be replaced by communication technology in order to avoid roundtrip unrealistic in a larger scale community. Furthermore, such a distributed approach requires technical means for code and documentation sharing and an integration environment with low entry barri es. Table I briefly shows the rather lightweight abstract schedule we pursued during our experiments. It should be noted that this process can be iterated. However, in the we held every four or six months. Here is likely to be subject to refinement or even re-placement in next phases of the ROLE project.

In the following, we provide details on the first iteration conducted during the ROLE Christmas Project.

All of the participants dedicated themselves to contribute components for an integration framework, individual learning services, either implemented or as mockup to reach the common goal of delivering an integrated PLE prototype.

After the collection of all contributions intended by the partners, we selected the ROLE Christmas Project Big Picture (cf. Figure 1). Given by the heterogeneity of the partners’ plans for contributions, we agreed on a common scenario serving as a concrete use case for a ROLE PLE prototype based on a Widget approach. For that purpose we chose a language learning scenario described in detail in Section III.

As a basis for ongoing documentation we decided to setup a document to be edited collaboratively by all partners, starting with the Big Picture, an elaborate description of the scenario and a time plan. Every partner added a description of his contribution and how it would fit with the scenario. Thereby, we did not require a perfect match, but at least a high degree of relevance.

A ROLE XMPP Server was setup for direct communication. A ROLE developer chat room was configured to log all group conversations on the server side. Thus, everybody could easily keep track of previous discussions, which turned out to be a helpful feature. However, restrictive firewall policies enforced by various partners in situations sometimes hindered the use of XMPP – a viable experience for future considerations regarding the use of our software (cf. Section IV.E.)

TABLE I. ABSTRACT PROCESS SCHEDULE

<table>
<thead>
<tr>
<th>Planning Phase (collaborative)</th>
<th>Development Phase (distributed &amp; independent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Start with of story-based use case scenario</td>
<td>• Develop &amp; document components</td>
</tr>
<tr>
<td>• Extract (non-)functional requirements</td>
<td>• Use own development environment</td>
</tr>
<tr>
<td>• Identify components</td>
<td>• Communicate with other developers</td>
</tr>
<tr>
<td>• Structure &amp; categorize components</td>
<td>• Continuously share code &amp; documentation</td>
</tr>
<tr>
<td>• Identify interfacing components</td>
<td>• Continuously run latest version of components in integration environment</td>
</tr>
<tr>
<td>• Agree on time schedule</td>
<td>•</td>
</tr>
</tbody>
</table>

Figure 1. ROLE Christmas Project Big Picture

Furthermore, in order to maintain the source code of all partner contributions, we agreed on utilizing a git-based [3] repository at github.com for reasons of wide visibility and acceptance in the open source developer community. Besides SCM functionality, github provides an issue tracker, Wikis, repository statistics, etc. All of these features were frequently used during the development phase.

With regard to a development environment for the individual part ner contributions, we defined the following simple policy:

1. All partners setup development environments.
2. One partner maintains integration environment and regularly pulls from the ROLE github.

During a later physical meeting we discussed a few options and suggestions regarding the choice of technologies as a basis for development and integration environments. The following considerations were taken into account:

1. How to quickly move forward and succeed with Christmas Project on schedule.
2. Avoid using technologies that will hinder us or force us to start over completely later on.
3. Make as few decisions as possible at this point in time.

Since the prototype should be Widget-based, we discussed a pre-selection of promising technologies for software components such as Widget engines, cont ainers, stores and repositories, inter-Widget communication mechanisms, protocols, etc. and started our developments after a first decision for one configuration.

All components of the system we used a technical foundation for our work. We promised to be platform and browser independent – unfortunately, this assumption was far from being fulfilled. It turned out that all regarded alternatives were still in an early experimental development stage. The consequence was an increased communication overhead among the partners for the purpose of finding, agreeing on a nåd changing to more acceptable solutions. Due to the fact that we could never rely upon the software components of our development and integration environments, continuous integration was barely possible, leading to the usual longer and error-prone final integration phase
III. REQUIREMENTS

In this section we will present an overview of the requirements elicited for the realization of the Christmas Project. We start with the language learning scenario we agreed upon as well as the underlying psycho-pedagogical model. We then continue with technical requirements for the realization of our prototype and considerations on how to fulfill them.

A. Scenario

In our language learning scenario, the learner, Tim, is an employee at Travel Books that sells books and videos on travel destinations. He works in the sales department and has to go to international fairs and to speak with distributors, bookshops and other business partners. As most business communication is in English, Tim needs to improve his English skills, especially in Business English.

One part of his learning strategy is to read texts and to learn its vocabulary using his PLE. For that purpose he adds three widgets: a Language Resource Browser, a Vocabulary Trainer and a Translator widget. All of them are visible on one webpage.

In the Language Resource Browser, Tim searches for a text and starts reading it. Each time he misses a word, he selects it and opens a context menu on it. The system then proposes him to either look it up in the Translator widget or send it to the Vocabulary Trainer widget (cf. Figure 2).

So he adds words that he considers as important to the Vocabulary Trainer and others he only looks up.

After reading the text, he has gathered a list of words he considers important to be repeated in future using the Vocabulary Trainer widget.

In the next days he contours reading new sparer articles regularly and his Vocabulary Trainer widget obtains more and more words. In an analogous manner he uses the Language Resource Browser widget to read and analyse the text and to learn the vocabulary. As he can use the word.

One day, he is learning with his Vocabulary Trainer, memorizing the words on the list and testing whether he knows them sufficiently well. He recognizes that he has problems to remember a certain word because he does not know the context anymore where it originally appeared.

Fortunately the Vocabulary Trainer always stores the link to the original text. So Tim clicks on the word and the original text appears in the Language Resource Browser widget and shows the sentence where the word was taken from.

Reading the sentence and thinking of the context facilitates him to memorize the vocabulary. Furthermore he improves his language proficiency by knowing situations where he can use the word.

The scenario can be extended by a group of learners, e.g., at universities that have an English Language course. The instructor sends them a list of newspaper articles that are available online. The students are asked to read and analyse the text and to learn the vocabulary. As they come from the same background (high school English level), they decide to jointly create a vocabulary list using the Vocabulary Trainer widget. Whenever a student finds an unknown word in the newspaper article, he adds it to the joint vocabulary list. The Vocabulary Trainer widget will also display words added by other students who have the same level in the English language. Words that have been added more often are sorted higher. Each learner in individually trains the list of words and the vocabulary knowledge. The group's average knowledge (number of words learned) is displayed in the widget, together with the current student's knowledge. Nevertheless it should be possible to deactivate the group functionality if the learner only wants to learn for herself.

B. Psycho Pedagogical Model

In this section we shortly present the psycho pedagogical model [4]. The central element of this model is a cyclic learning process model consisting of several learning phases related to learning activities. Following the connections between learning phases, activities and tools, a sequence of learning tools can be derived. Furthermore, two different kinds of learning tools are identified, i.e., self-regulated learning tools that can be used in a self-regulated learning process and tools used for self-regulating the own learning process.

A navigation tool guiding the learner through a self-regulated learning process by means of providing learning activities and tools was proposed. The partners created an interactive mock-up of such a navigation tool (cf. Figure 3) and presented it in the context of the scenario from the previous section.
The contribution of education professionals had a great impact on the technical developments and showed us once more that technical and conceptual work should be conducted hand-in-hand.

C. Technical Requirements

For the technical realization of a first ROLE PLE prototype, we decided to follow an approach of intercommunicating Widgets. The expectation from choosing such an approach were relatively loose coupling between individual widget parts. It should not be that we do so experimented with such an approach, because in the future projects, developers outside the consortium should be enabled to work completely independent from other developers.

First, a technical infrastructure was needed as a basis for our prototype. Starting from the Big Picture, we identified requirements for the following components of such an infrastructure:

1. Widget Container/Engine
2. Widget Store/Repository
3. Widget Interoperability Mechanisms
4. Widget User Interface

As part of experiments with Apache Shindig [5], the reference implementation of an OpenSocial [6] container currently under incubation at Apache, OpenSocial defines a common API for social applications across multiple websites and also includes a specification for widgets – gadgets in OpenSocial terminology. As an alternative we considered SocialSite [7], based on Glassfish and Apache Shindig with the ability to run OpenSocial gadgets and have them backed by the same social graph. Furthermore, Apache Wookie [8] was taken into consideration as a solution for adding W3C Widgets [9] as well as OpenSocial and Google (Wave) Gadgets to web applications.

1) As solutions for a Widget store/repository, we considered the following four solutions:

- No store/repository, fixed list.
- Wookie
- Google Gadget directory
- ROLE widget store (remains to be built)

A fundamental requirement to Widget interoperability was that the supert port of OpenSocial/Container communication. There should be no major configuration needed on the side of end users assembling widgets. Furthermore, it should be easy to build containers for the chosen widget technology. Integration into existing systems, like LMS should be possible. Preferably, standard content creators should also support these technologies. The following technologies were taken into closer consideration as potential candidates:

- Gadget pubsub [6]
- OpenAjax Alliance hub2.0 pubsub [10]
- Open Application (draft) [11]
- HTML5 Data [16]

2) Regarding a Widget user interface a couple of different approaches were considered, e.g. portal pages such as iGoogle, Wiki or LMS approaches, etc. However, we decided to keep things simple first using a fixed HTML page. An additional solution experimented with was the integration in Graasp (http://graasps.epfl.ch), a Web 2.0 contextual aggregator of people, spaces, assets and tools, in order to be able to make use of its built-in mechanisms for sharing, commenting, and recommendation.

For the first experiments, we decided to use a configuration of SocialSite as a container, no widget store, simple HTML page or Graasps as user interface and OpenApplication based on Gadget pubsub for inter-widget communication. Further, hermore, the previously described R OLE XMP Server was used for experiments on remote inter-widget communication. For the first prototype, the primary focus was thus put on the development of communicating widgets.

IV. PARTNER CONTRIBUTIONS

In this section, we will present the contributions of all partners for the Christmas Project prototype. The first two contributions are targeting at an integration framework focus on the communication between widgets. All remaining contributions consist of different types of widgets either especially for the language learning scenario or with rather general functionality.

A. Inter-Widget Communication

Uppsala University introduced the Ope n Application Event API to provide a generic solution to the requirement of inter-widget communication in the scenario. The most important aspect of the solution is that widgets need not be “hard-wired” against each other. Instead, they communicate using well-known data expressions, with the intention that widgets will understand the parts that are important to them.

The basic principle behind the event API is that all widgets are notified of all events. No specific subscription step is necessary. All widgets are given the opportunity to react to any event, which the y may choose to do depending on event type, message type, message content, etc.

1) Events types

The event type includes: state, load, modify, save, select, unselect, startDrag and stopDrag. A state event is different from the other in that it has no relation to the event resource, and therefore no resource is sent along. Instead, a state event indicates a change of state in the widget that sent it.

2) Event Structure

Notifications of events are sent out as messages. The message consists of the relevant resource, if any, depending on the event. The message is wrapped in an envelope containing further event information.

- event - the event type (see previous section).
- type - the message type
- message - the message, for example a resource.
- uri - the message's URI, if any.
- date - timestamp of when the event occurred sharing - How the event is allowed to be used.

At the moment, the event API consists of a Gadgets PubSub channel, in which the messages are published and thereafter sent out to all widgets.

An Open Application compliant widget gets subscribed to the PubSub channel when the widget is loaded. The shar-
ing property is intended to specify how the data may be used: on the same page, only on the user’s machine, by a service under the user’s control, by participants with access to the same widget instances, or that it may be transmitted to various services. Each level includes all the privileges of the previous levels.

3) Message Types

A number of message types are defined. These include: namespaced-properties, JDL, JSON, URL, HTML, XML and MIME content. Of these, namespaced-properties is intended for simple RDF-like metadata with direct properties, and MIME content for unparsed text or binary data.

B. Web 2.0 Platform for Collaborative Organization of Information and Tools

The EPFL team developed a Web 2.0 platform, namely Graaasp, to help users to collaboratively organize information and tools toward a given goal or activity. Tools in Graaasp are implemented as widgets. In addition to the standard add, remove, browse, group and share operations, Graaasp also supports tagging, rating and commenting. The widgets are imported/bought from a widget store and linked to the given Graaasp space dedicated to a learning activity.

Once the activity is configured the user can switch the view to play with instances of selected widgets. The widget instances are rendered in a widget container managed by a Wookie engine. Any widgets following the W3C widget specifications can be instantiated into Graaasp.

Based on both the created activity structure and the user ratings, a recommender system that would contain externally recommend widgets to users is being developed. Similarly, Trust and Reputation algorithms for widgets are also considered.

C. Monitoring of User Behaviour

To provide recommendation and self-evaluation mechanisms Fraunhofer FIT developed a CAM [17] widget to unobtrusively monitor user behavior. Other widgets, like the vocabulary trainer widget (cf. next section), trigger events on different user actions which are then broadcasted to other widgets using Open Application. The CAM schema provides a standardized data format to store user activities and thus foster widget interoperability since every widget could access these data.

Since the CAM widget is a simple subscriber widget, it listens to every event published from any other widget and collects the data. The collected events are then transferred into the CAM schema and afterwards stored in a database. As all control should be with the user, she can decide between different storage modes. For the Christmas project we therefore specified three different storage modes which can be selected by the user (cf. Figure 5).

If the user decides to store her activities remotely, the CAM widget transfers the data to a central CAM repository, where all events of every user are stored. After creating a CAM instance, the CAM widget calls a Web service [18] passing the CAM information which is then stored in a database. The local storage mode uses the Gears plug-in [19] to store CAM information in a local database. The Gears plug-in is available for many platforms and supports all common browsers. It provides a SQLite interface to easily create a database and store information inside of the local browser profile. An alternative approach to Gears is using HTML5 [16]. Since the specification of HTLM5 is not finished yet and is currently not supported by every browser we have chosen Gears. If the user does not want her usage behavior to be monitored she can choose the storage mode off.

To generate recommendations based on CAM, these can either be based on the user’s own previous behavior or the usage history of others can be taken into account as well. The different storage modes have effect on the generation of recommendations. In the remote mode the user can get recommendations, but also allows the system to use his information to generate recommendations to other users. The local storage mode only allows retrieving recommendations, and disallows the system to use her information to generate recommendations to other users. The user does not want her usage behavior to be monitored she can neither get recommendation nor support the system to generate recommendations to other users.

As we had no Vocabulary Trainer CAM data for the Christmas project, we transferred some PLE monitoring data [21] into the CAM schema and provided a separate Web service which offers methods to generate recommendation and self-evaluation statistics.

D. Visualization of Monitored Activity

KU Leuven developed a dashboard that enables students and teachers to monitor learning activities. In the dashboard students can monitor the progress they made...
on a certain course or task and compare themselves with other students working on the same task.

An important feature of the dashboard is learning material recommendation. Based on the learning material other students have used, who have progressed further, we can recommend interesting learning material to the student. A student can compare his activities with other students and a teacher can get a general overview of what is going on in the course, see if it meets expectations and detect potential problems.

Figure 9 illustrates the first version of the user interface for our application. The student can select the course and will be presented with 2 different charts, a course analytics overview and document recommendations. Every line in the chart in figure 1 is a student. The chart shows when the student worked (horizontal axis) and how long he worked on the course (vertical axis). The red line shows a selected student, we see that the student was very late in finishing the course and that he spent a lot of work in a small number of sessions. This view enables a student to compare his progress with that of his fellow students. Another visualization uses parallel coordinates [22]. It shows a set of metrics on parallel axes. A student is represented as a polyline with the vertices on every axis. The metrics are: the average and total time spent on the course, the number of documents used and the average time of the day that a student works. By visualizing these metrics next to each other one can grasp another view on the course activity and discover trends.

The widget is developed in Adobe Flex. To monitor user activities, we use CAM [17]. In order to test the tools, we used course data provided by U&I Learning [21]. Together with our partners at FIT, we experimented with the data to natively build a community visualization widget. This tool could allow students to find fellow students who have the same language proficiency and to chat or collaborate with. The widget would communicate with the chat module of RWTH to provide chat functionality. We are currently implementing this widget.

**E. XMPP Chat Widget**

The contribution of RWTH Aachen University is a Chat Widget providing a simple Instant Messaging (IM) client (cf. Figure 7) based on the XMPP [12][13] protocol. The widget offers interface elements for 1-on-1 conversations, the management of buddy lists and user presence information. In its default configuration, the widget connects to the ROLE XMPP Server. However, connections to arbitrary XMPP Servers are possible. Regarding the ROLE Christmas Project’s language learning scenario, the XMPP Chat Widget contibutes to synchronous communication between learners integrated in a PLE. The widget is as a rich source for communication events, e.g. updated presence information, incoming/outgoing messages, etc. The integration of event publishing into the Open Application Approach to be captured by CAM is planned for the near future.

Besides the added value of XMPP chats between learners and basic communication statistics, these two widgets demonstrate how inter-widget communication could be realized between remote widgets (via sending messages over XMPP) and between local widgets (e.g. via a Gadgets pubsub). There already exist specifications on XMPP Extension Protocols (XEPs) for Publish/Subscribe [14] or Personal Eventing Protocol [15] mechanisms, which will be side red for a seamless remote/local inter-widget communication for future developments.

**F. Language Learning Widgets**

In order to fulfill the requirements of the language learning scenario, imc AG developed the English learning widgets: The Language Resource Browser Widget, the Vocabulary Trainer Widget and the Translation Widget. These widgets demonstrate a reasonable use of inter-widget communication by sending a node receiving terms described by a term and its context and source.

![Figure 6. Dashboard with line chart](Image)

![Figure 7. ROLE XMPP Chat Widget](Image)
COLLABORATIVE DEVELOPMENT OF A PLE FOR LANGUAGE LEARNING

1) Language Resource Browser Widget

The Language Resource Browser widget (cf. Figure 8) allows users to consume media and send terms to other widgets processing the information. Examples of such widgets are the Translator Widget where the term will be translated or the Vocabulary Trainer where the user can add this term to a vocabulary list. At the moment the widget offers three different tabs. The "Text" tab works like a web browser. It displays a page at a given URL in an iframe where the user can select the term and context. The source of such a term item will be the URL from the page.

In the second tab "Own Text" the user can add her own text taken from an online or offline resource. The third tab provides support to browse for different media such as video and audio. While watching or listening to the media, the user can enter a term in a field. The source of such a term item will be the URL from the media and the context will be defined as "Media Context".

2) Translator Widget

The Translator widget allows a user to translate terms or sentences. It translates either a term which was entered by the user or a received term item. We combined different Web services (i.e. Wikipedia, Google Dictionary, DICT.ORG, Google Translate) for the translation process. At the moment only English to German is supported, but the language pool could be extended to all languages supported by the services above.

3) Vocabulary Trainer Widget

The Vocabulary Trainer (cf. Figure 9) widget is implementing a slightly modified Leitner system [23]. A vocabulary list consists of five different buckets. If an item is added it will be put in the first bucket. If the user is training a list and knows the right translation the item will be moved to the next bucket and else it will be moved to the previous bucket.

The information is stored on a central server and accessed using REST Web services. Each user has a unique login and authentication is done by basic access authentication over REST. For translation the same Web services are used as in the Translator widget, and Flickr is used to suggest pictures for term search. Vocabulary items are stored in a list which can be managed by the user.

The widget has four functionalities represented by four tabs: "Add", "List", "Train" and "Stats".

The "Add" tab allows users to manually enter a new term or sentence. The context of that term and its source will be displayed. The user can get help by viewing the source of that item or viewing the image to that item (if there exists one). The fourth tab "Stats" shows statistics of the training. It displays a global score and a score for each list.

G. Federated Search & Language Processing

In traditional educational scenarios, teachers typically provide appropriate learning materials and give feedback on student essays. Vienna University of Economics & Business contributed two widgets which are useful for these purposes and indicate their application for language learning.

ObjectSpot, a widget for federated search of academic papers in different digital libraries, has its origins in the iCamp project (http://www.icamp.eu). This search client allows plugging in different digital repositories, whereby the documents are retrieved via the Simple Query Interface (SQI) [24] standard. The central core of this widget is the ranking algorithm which has been developed over several iterations and mixes in the search results from the repositories on the fly [25]. In practice, this widget is useful for both learners and teachers to retrieve the most appropriate literature for a specific knowledge domain. By default, the most important digital libraries count aing...
COLLABORATIVE DEVELOPMENT OF A PLE FOR LANGUAGE LEARNING

academic papers, e.g. ACM, IEEE, Google Scholar, CiteSeer, EBSCO, etc., are included.

The screenshot shows the results for the query term 'open responsive learning environments'. At the top, the search term and the state of the repositories are displayed. At the bottom, the user can navigate through the pages.

On the left hand side, a user can 'lock' appropriate results, thus giving explicit relevance feedback and recommendations for others. This mechanism can also be used to export search results through a feed-based API if another tool is attached to ObjectSpot. Pressing the option button, a user can configure SQI-enabled digital repositories for her search client, visualize the location of the repositories on a map, get statistics on the quality of the repositories, export the results as RSS-feed, get recommendations for a query term, or plug another tool to ObjectSpot.

Conceptalyzer comprises a language processing widget which builds upon a LSA-based Web services developed within the LTfLL project (http://www.ltfll-project.org).

This widget analyzes online resources (e.g. Wikipedia articles or RSS feeds) in terms of the concepts behind the text and visualizes them according to their relevance. Application areas of this widget comprise learner positioning, monitoring one's conceptual development. It can also be helpful for teachers in preparing learning materials or grading students [26].

The screenshot shows the result of the analysis of a Wikipedia article, whereby the relevant terms are visualized in the form of a 'concept cloud'. The size of a term indicates its relevance for the article while the color links to the text corpora used to train the LSA function.

V. DISCUSSION OF WIDGET TECHNOLOGIES

During the development process of the ROLE Christmas Prototype PLE, we collected a set of valuable however negative experiences to be shared with other developers working with the technologies we attempted to combine. These experiences will be discussed in this section.

One of the most surprising experiences from the developer perspective was the immature state of many widget technologies, especially with regard to inter-widget communication, one of our main requirements for the language learning PLE. During the development process we tested the following three OpenSocial compliant Gadget development environments:

1. Apache Shindig
2. SocialSite (based on Shindig within Glassfish)
3. OSDE (Eclipse Plugin with integrated Shindig)

With all of the above systems we encountered at least one of the following problems:

- Lack of Forward Incompatibility
- Client Browser Dependence
- Server Platform Dependence
- Inaccessible Bugs in Generated Code
- Incompatibility with External Libraries
- Lack of Developer Support

The first problem was related to the installation of a gadget container, in particular with SocialSite. First, the current SocialSite distribution is restricted to specific, already outdated versions of Glassfish and Shindig, and thus is not forward compatible - an essential property, when working with experimental systems. Given the diversity of devices and platforms available to the developers, we quickly had to find out that platform and browser independence was not given at all. Container-side or browser-side errors were the result. The most essential problem was the inaccessibility of bugs in JavaScript code. In many cases, problems occurred outside the source code under developer control. The reason was a malfunction in the code production performed by the container itself. Further more, error messages were cryptic and incomprehensible and thus did not provide any hint to the original location of an error. Furthermore, we lost a lot of time communicating possible alternatives. An excursion to the usage of Apache Shindig instead of SocialSite was also not successful for all of us. Further problems were related to the incompatibility of external JavaScript libraries with the Gadget container, which is a result of strange code rewrite-features. Especially with regard to JavaScript libraries, we had to experience that these libraries were not far enough for the realization of our goals and definitely need improvement. Finally, we had to experience that the developer support by SocialSite team was not available at all. At the time of writing this document, it seems quite obvious, that SocialSite is dead.

We finally managed to deploy our prototype in Graaasp in a rather stable version, but still with a lot of open issues.

![ObjectSpot Gadget](http://www.i-jet.org)

Figure 10. Objectspot – Federated Search Widget

![Conceptalyzer Gadget](http://www.i-jet.org)

Figure 11. Conceptalyzer – Language Processing Widget
to be tackled in later development stages of the ROLE project.

Drawing the conclusions from our experiences, we can state that the technologies we experimented with were insufficiently mature for the deployment of a stable integrated prototype assembled from a set of innovative tools realized using different technologies. For further collaborative distributed development experiments we agreed on a short, but regular biweekly meetings in order to get aware of occurring problems earlier. The agenda will be inspired by action items of W3C meetings.

VI. CONCLUSION & OUTLOOK

In this paper we provided a report of the collaborative distributed development of the ROLE Christmas Project resulting in a prototype of a Widget-based PLE for language learning. We first described the development process condoned among different partners from both academia and industry, with varying technical backgrounds, motivations and interests regarding the whole project. We pointed out that regular communication and the clear definition of goals and a schedule was inevitable during the whole process. Furthermore, we listed useful technical means of collaboration such as communication media, shared document entainment, shared code repositories, etc. Furthermore, we draw the conclusion that our approach did not work as expected, raising the necessity for an improved approach better suited for collaborative distributed development of Widget-based PLEs. In a section on requirements we present our use case scenario and gave an insight into the conceptual and technical work must happen together. We elicited technical requirements from a basic infrastructure for distributed PLE development and present a set of solutions for its realization as a foundation for our experiments. We then provided an overview of the innovations resulting from individual parts and collected, ranging from integration technologies to scenario-dependent and independent learning services in web sites. Finally, we critically discussed the outcome of the ROLE Christmas project and reported a set of technical issues hinting to the conclusion, that Widget technology is not mature and stable enough to enable distributed collaborative PLE development without hassle at this point in time. However, we worked out the requirements and associated problems for distributed implementation of Widget-based PLEs and collected a lot of valuable experience that will shape future endeavors. In an upcoming completion solidification phase, the developer team will stabilize the core service, improve an and align the development process and then continue work towards a number of bundles for the implementation of the ROLE test bed scenarios.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7 2007-2013) under the grant agreement no 231396 (ROLE project).

REFERENCES


AUTHORS

D. Renzel (re nzel@dbis.rwth-aachen.de) is with the Chair for Computer Science 5, RWTH Aachen University, Aachen, Germany.

C. Höbelt (christian.a.hoebelt@im-c.de) is with imc information multimedia co mmunication AG, Saarbrücken, Germany.
D. Dahrendorf ([daniel.dahrendorf@im-c.de]) is with IMC Information Multimedia Communication AG, Saarbrücken, Germany.

M. Friedrich ([martin.friedrich@fit.fraunhofer.de]) is with the Fraunhofer Institute for Applied Information Technology, Sankt Augustin, Germany.

F. Mödritscher ([fm.ödrit@wu.ac.at]) is with the Vienna University of Economics and Business, Vienna, Austria.

K. Verbert ([katrien.verbert@cs.kuleuven.be]) is with the Katholieke Universiteit Leuven, Leuven, Belgium.

S. Govaerts ([sten.govaerts@cs.kuleuven.be]) is with the Katholieke Universiteit Leuven, Leuven, Belgium.

M. Palmér ([matthias@nada.kth.se]) is with the Uppsala University, Uppsala, Sweden.

Evgeny Bogdanov ([evgeny.bogdanov@epfl.ch]) is with the École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland.

This article was modified from a presentation at the International Conference of Interactive Computer Aided Learning ICL2009, September 2009 in Villach, Austria. Submitted 11 September 2009. Published as resubmitted by the authors on 17 January 2010.