D2.4. Building a Personal Learning Environment with Language-Technology-based Widgets: Services v2 - integrated thread

Language Technologies for Lifelong Learning

LTfLL -2008-212578

Project Deliverable Report

Deliverable D2.4. Building a Personal Learning Environment with Language-Technology-based Widgets: Services v2 - integrated thread.

Work Package	2
Task	6
Date of delivery	**Contractual**: 31-10-2010 **Actual**: 16-12-2010
Code name	D2.4 **Version**: 1.0 **Draft** **Final** ☐
Type of deliverable	Report
Security (distribution level)	Public
Contributors	Bernhard Hoisl, Debra Haley, Fridolin Wild, Lucas Anastasiou, Katja Buelow, Robert Koblischke, Gaston Burek, Mathieu Loiseau, Thomas Markus, Traian Rebedea, Hendrik Drachsler, Helmut Kometter, Eline Westerhout, Vlad Posea
Authors (Partner)	UKOU, WUW, OUNL, BitMedia, UTU, UPMF, PUB-NCIT, IPP-BAS
Contact Person	Fridolin Wild
WP/Task responsible	WP2
EC Project Officer	Ms. M. Csap
Abstract (for dissemination)	This deliverable reports on the results achieved by the LTfLL work packages in their efforts toward interoperability of the LTfLL tools and services. There are two aspects: one is the pedagogical utility of achieving interoperability; the other aspect involves the technical features. The technical basis of the interoperability is to use Wookie widgets in Elgg and is thoroughly described here. Finally, the deliverable provides details and screen shots of each widget for each LTfLL service embedded in the Elgg environment.
Keywords List	Long thread, short thread, PLE, widget, Wookie, Elgg

LTfLL Project Coordination at: Open University of the Netherlands
Valkenburgerweg 177, 6419 AT Heerlen, The Netherlands
Tel: +31 45 5762901 – Fax: +31 45 5762800

Table of Contents

Executive Summary

This deliverable reports on the results achieved by LTfLL in its efforts toward creating interoperability for the developed tools and services. There are two aspects: one is the pedagogical utility of achieving interoperability; the other aspect involves the technical features.

The 'long' thread, as described in D3.3 and slightly modified by the consortium in September in Bucharest, was devised as a demonstration of pedagogical improvement resulting from combining four of the services into one personal learning environment (PLE). This scenario involves an English speaking learner in the IT domain answering the question "What is Web 2.0?". The learner uses the iFLSS to locate potential learning resources. Then, the learner uses PenSum to write (and improve) a synthesis. The completed synthesis is fed into Conspect, which provides a conceptogram and a list of concepts. If the user clicks on one of the concepts, Conspect automatically passes the concept to iFLSS, which subsequently provides learning resources from sources such as YouTube, delicious, or BibSonomy. A tutor then picks a few concepts and asks the learners to discuss them in forums or chat groups. Finally, PolyCAFe helps in analysing this discussion. The 'short' thread (ST) involves LeaPos and FLSS. The tutor adds missing lexicalisations, which are then stored by FLSS and used later by LeaPos for an improved and automated annotation of learning materials and answers.

The pedagogical effectiveness of the long thread will be evaluated and reported on by WP7 in D7.5.

The technical basis of the interoperability is to use Wookie widgets in Elgg (see D2.1, D2.2, and D2.3). For this idea to become reality, several modifications needed to be made to existing open source software. The existing services needed to be redeveloped as Wookie-compliant widgets. The interwidget communication (IWC) facilities created had to be aligned with the Wookie open source community. A new version of the Elgg plug-in for Wookie widgets had to be developed. A project wide widget template was created to provide a consistent look and feel to the widgets. Finally, an additional widget server (for OpenACS) had been developed to further exploitation of the project results into another large open source community. These technical details are covered fully in this deliverable.

The deliverable provides details and screen shots of each widget for each LTfLL service embedded in the Elgg environment.

1. Introduction

Deliverable 2.4 documents how the various services of LTfLL have been combined in a personal learning environment (PLE), which is a network of people surrounding an individual with the people in this network making use of artefacts and tools while they are involved in isolated or collaborative activities of more or less planned (co-) construction of knowledge and information. The individual at the centre of the PLE actively and passively modifies this environment through actions with the intention to positively influence her social, self, methodological, and professional competence, i.e. changing her potentials for future action. Though the individual tries to structure the environment, she is not fully in control to design it, as characteristics and affordances of and relationships between the agents in the network (people, tools, artefacts) are not oriented towards a common goal and according to a joint plan.

Although PLEs were first conceived in opposition to virtual learning environments and learning management systems, today they have become rather a new technique and research stream. Its focus is on allowing for the more flexible recombination of learning tools, populating them consciously with content and in social networks, and mixing them with elements of the rest of the personal environments surrounding and supporting learners beyond their learning tasks so that the environment is truly 'owned' by the user.

Besides empowering the users to more consciously co-design their environments according to their needs, this idea is expected to bear several other advantages: it helps to blend formal, non-formal, and informal learning, education, and work (cf. Sporer et al., 2007); it creates opportunities for the development of rich professional competences such as digital literacy and media competency (Wild et al., 2009); and it provides a way to cope with the distributed nature of educational resources in a global information society.

Technologically, this new technique is rooted in a software development tradition called opportunistic design, which focuses on rapid application development through the maximisation of code re-use and re-appropriation of soft- and hardware components (Hartmann et al., 2008). Popularly, these approaches are also called 'mash-ups', 'gluing', or 'wiring'.

PLEs now take this idea of re-use and re-appropriation onto the next level. Through the deconstruction of learning tools into use-case sized widgets, through the provision of plug & play auto-configuring middleware services, through standardised data sources, and with the help of bootstrapping tools with high usability, end-users can be facilitated in building up a personal learning environment adapted to their needs.

This deliverable reports on the efforts within the LTfLL project to develop a personal learning environment approach and prototype. It is about the enhancements of a PLE runtime environment based on the open source project Elgg. It is about research in and development of interwidget communication components that allow for the aligned operation of basic building blocks. And it is about the widgetisation of the various services developed within the project. This deliverable is therefore organised into two parts. First (Section 2 to 4), the pedagogically sound use of the services developed – combined in a 'thread' – is illustrated. The subsequent

section 5 then documents the technical implementation of the runtime environment based on Elgg, the interwidget communication (IWC) component, and relevant design decisions.

There are two threads presented: a 'long' thread (long thread) (Section 2 and 3) demonstrates the PLE approach along a workflow that involves a large arrangement of widgets, whereas the 'short' thread (ST) (Section 4) involves a smaller arrangement of widgets. Each thread is a sequential set of actions comprising a pedagogically sound learning scenario and involving a particular arrangement of widgets, which – used together – facilitate the achievement of an overarching task.

Depending on the actual topic chosen or assigned to the learners, they first receive suggestions of learning resources from the iFLSS (inFormal Learning Support Services) in this long thread. They use PenSum to create a synthesis of their research. PenSum feeds the synthesis into CONSPECT, which provides a list of concepts relating to Web 2.0. Then, iFLSS will provide learning resources gathered from social networks for the extra concepts. A tutor picks some of the concepts and directs the learners to chat about their common understanding of the term. Finally, PolyCAFe analyses the chat.

The relationship of this deliverable to previous work package deliverables is as follows. D2.1 outlined the initial concept, D2.2 documented the pilot services, and D2.3 documented the first release of the service prototypes and provided information on the integration facilities and guidelines that paved the way for the deep integration of the final version of the services, i.e. the implementations demonstrated in the scenario of the long and short threads.

More details on the pedagogical scenario of the threading of the services including alternative arrangements, a set of alternative scenarios, and a more detailed description of this long thread shown within this deliverable can be found in deliverable d3.3.

2. Arrangement of the LTfLL widgets in a 'long' thread

The purpose of the 'long' thread for this deliverable is to demonstrate how a larger set of LTfLL tools can be integrated by learners into an *interoperable* personal learning environment. This means, that the involved widgets, services, and data sources have to collaboratively exchange data so that they facilitate the successful accomplishment of an overarching task that would not be possible when using the involved tools in isolation (Wild & Sobernig, 2006).

Technically, this involved realising interoperability along several dimensions, as outlined in Palmer et al. (2010, to appear): distinct features for creating interoperability along the dimensions screen, data, temporal, social, activity, and runtime have been conceptualised and implemented within the project. This particularly involved the enhancements of the widget server Wookie and its interface to the runtime environment Elgg with extensions of presentation and layout mechanisms and the introduction of interwidget communication facilities (see D2.3 and updates following in Section 5).

As for a prototypical overarching task, Figure 1 depicts the 'long' thread selected for this demonstration. The scenario takes place in a formal learning environment in the IT domain in the English language. The particular task is: Explain the meaning of the term 'web 2.0'.

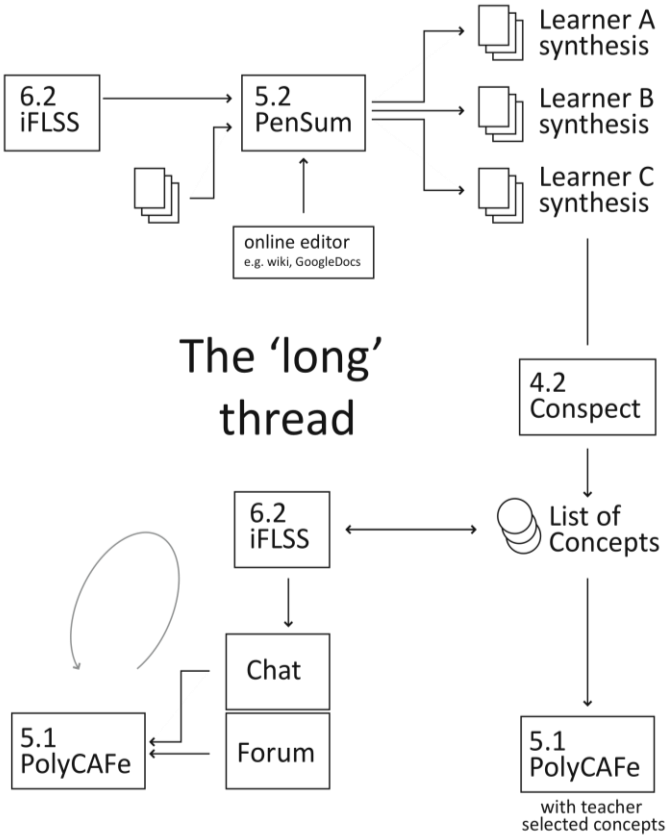


Figure 1. The long thread.

Students therefore first ask iFLSS (T6.2) for appropriate learning resources. Each student produces a synthesis of his understanding of the resources provided, either directly in Pensum (T5.2) or using alternative online editors such as a Wiki or GoogleDocs. The learners use PenSum (T5.2) to gather first feedback on how well they are doing with their synthesis of the resources about 'web 2.0'.

The user submits then the synthesis to Conspect (T4.2). Conspect produces a list of concepts including those the user covered, those the user should have (but did not), and those extra. Per mouse click they can have iFLSS provide additional learning resources gathered from social networks for any concepts of interest, particularly for the ones missed out on.

Learners and tutors can then select a number of concepts to be discussed by clicking on them in Conspect – say six, for example. The learners will discuss these concepts and potential misunderstanding using a forum or a chat. They will be able to request learning material from iFLSS and the discussion will be analysed by PolyCAFe.

Long thread interactivity

		Conspect	PolyCAFe	PenSum	iFLSS
		4.2	5.1	5.2	6.2
Conspect	**4.2**		concept list to 5.1	accept syntheses in the form of RSS feeds	provide links on list screen so user can get docs from 6.2
PolyCAFe	**5.1**	accept concepts from teacher and/or Conspect		none	none
PenSum	**5.2**	provide RSS feeds that Conspect can access	none		accept input from 6.2
iFLSS	**6.2**	accept search terms from Conspect	accept search requests from chat/forum	send info to PenSum	

Table 1. Service interaction matrix of the long thread.

There are several points of intersection between the individual services in this long thread. They are shown in Table 1 as an interaction matrix. The tasks involved are listed in the appropriate cell. Reading across the first row, T4.2 will send concepts to T5.1, will accept syntheses in the form of RSS feeds from T5.2, and will provide links on the list screen so the user can get documents from T6.2.

The long thread documented in this deliverable is just one possible combination of the LTfLL services. Appendix C shows yet another possible arrangement of widgets that could suit a particular learner.

Figure 2 shows a portion of the long thread widgets working together in Elgg. In this example, Conspect provides a list of concepts. The user has selected the term, 'CSCW'. This term automatically appears in the iFLSS search widget followed by potentially relevant learning resources in the other three widgets.

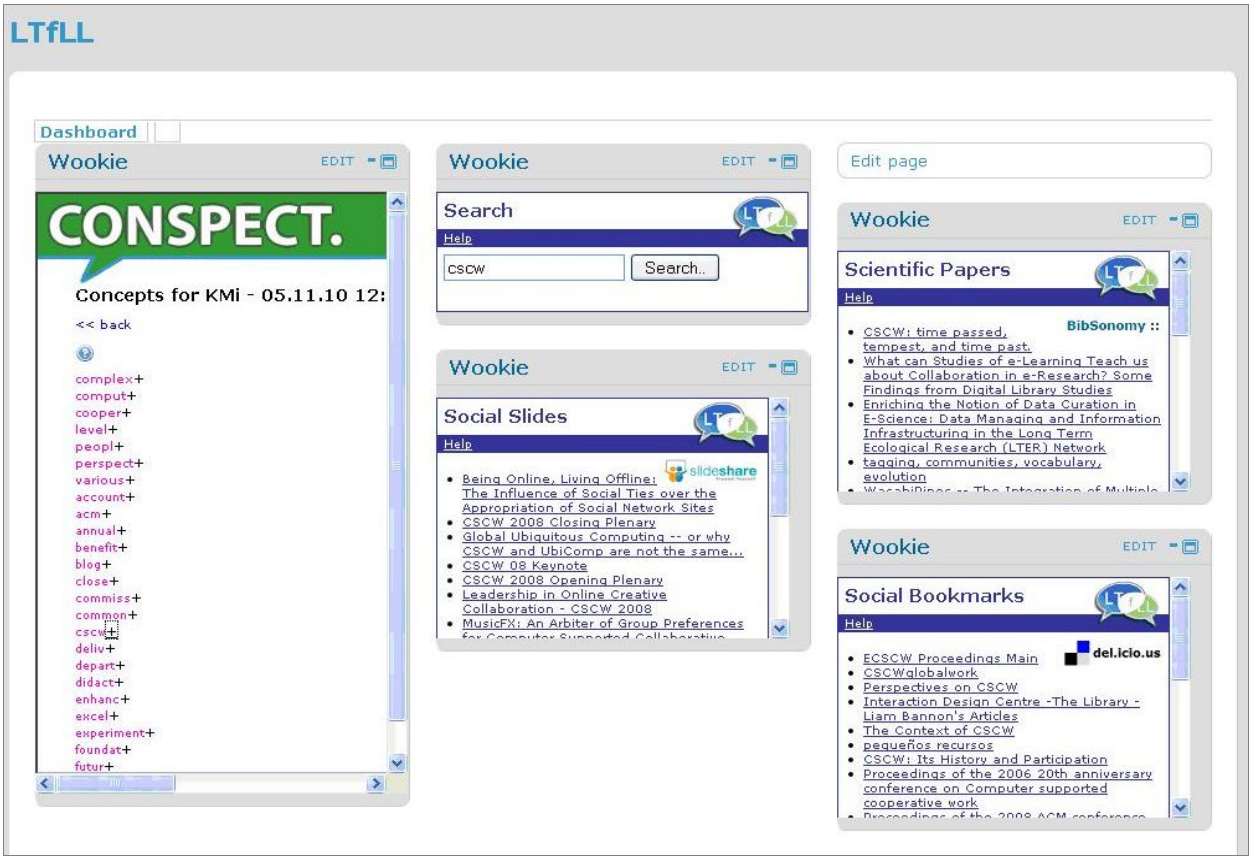


Figure 2. An example of some long thread widgets interacting.

3. Widgets in the 'long' thread

In the subsequent section, the widgets involved in the task sketched out in the long thread are described in more detail. This description is complemented with more technical details in the Annex A (needed for developers to take up project results or to interface with other building blocks of personal learning environments).

3.1 Conspect (T4.2)

Figure 3 shows the two CONSPECT widgets working together. The user enters the term, 'bbc', in the feeds widget, which lists appropriate feeds. If the user clicks on 'add', the feed is added automatically to the main Conspect widget. Appendix B describes the cross-domain IWC techniques used by Conspect.

Conspect widget

The main Conspect widget (shown in Figure 3) is where most of the functionality takes place. The user requests a feed from which CONSPECT creates a conceptogram and a list of concepts. The functionality has been thoroughly documented in D4.3.2.

Feeds widget

The feeds widget was added to demonstrate how various functionalities of Conspect can be removed from the main widget and placed in another, related widget.

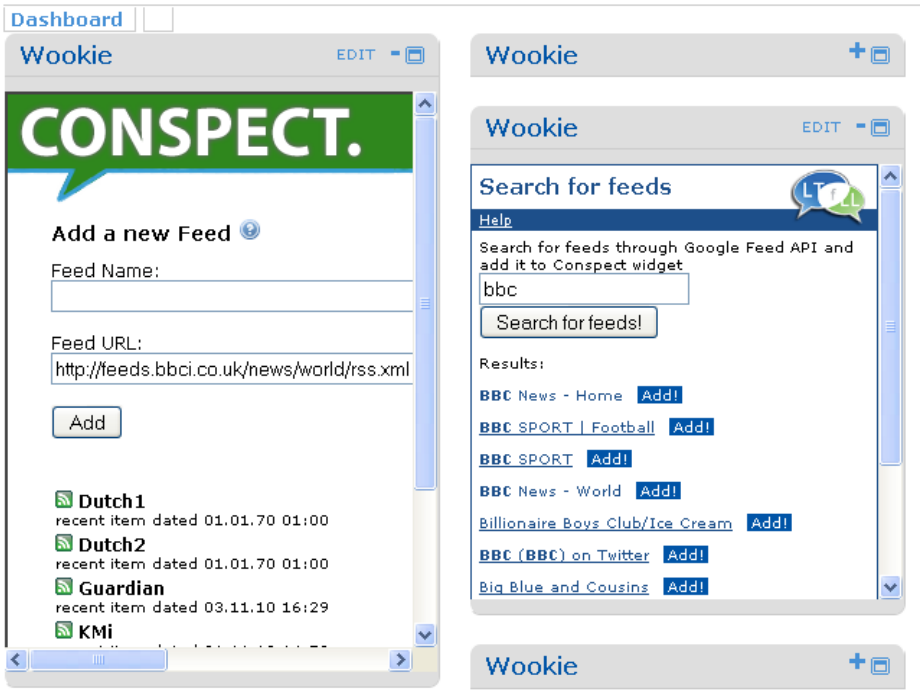


Figure 3. Conspect widgets.

3.2 PolyCAFe (T5.1)

As mentioned in d5.3, PolyCAFe has seven important widgets divided into: maintenance widgets available only for tutors and administrators, and feedback widgets that are used by both learners and tutors.

The maintenance widgets communicate with PHP-based services using JSON, while the feedback widgets take advantage of the results offered by a Java-based AXIS2 web service and they are provided using XML (for large quantities of data) and JSON (for smaller updates). Further information on this topic can be found in D2.3.

The two maintenance widgets are: assignment management and conversation management; while the five feedback widgets have been divided into: conversation feedback, conversation visualization, participant feedback, utterance feedback and the enhanced search conversation widget.

All these widgets are presented in the next subsections, starting with the feedback widgets that provide the most important functionality for the users. Moreover, all the widgets are Wookie compatible and have been deployed into Elgg.

Conversation feedback

The conversation feedback widget (Figure 4) permits selecting the assignment and conversation that the user is interested in, by using JSON and the maintenance services. After the conversation is selected, the widget processes an XML file returned by the feedback service that contains the whole conversation, plus the automatic annotations made by PolyCAFe. It then displays to the user several pieces of information of interest for the whole conversation, including:

- The list of most frequent topics in the discussion.
- The list of relevant concepts within the actual discussion – these are determined by comparing the concepts in the conversation with the relevant (important) concepts in the semantic space configured for the current assignment.
- The list of similar topics with the entire discussion, which are not mentioned in the discussion. These topics are present in the semantic space used for the assignment, but have not been used by the participants of the conversation.
- The average score for an utterance. It should be as high as possible: a value between 6-7 identifies a good conversation, a value lower than 6 indicates a conversation that could have been better, and a value greater than 7 denotes a very good conversation.
- Information on collaboration that contains indicators that should have a value as high as possible because the explicit, implicit links are very important, as well as personal opinions, speech acts, arguments, etc.

Figure 4. Conversation feedback.

Conversation visualisation

This widget (Figure 5) was described in d2.3; therefore this update is very brief. The main functionality of this widget is to determine how good the collaboration and interaction is within a conversation. It also allows the user to have a better view of the conversation in order to find the most important areas from the collaboration and participation (engagement) points of view. In the upper part of the screen, there is the conversation graph with utterances as nodes and the implicit and explicit links as edges. In the middle part of the screen, the collaboration graph is displayed.

In the lower part of the screen you have three tabs with controls:

- The first tab, "Options", contains the options for the conversation graph visualization.
- The second tab, "Conversation thread", permits clicking on an utterance in the conversation graph and highlighting the discussion thread that the utterance is part of.
- The third tab, "Special threads" allows the users to inspect the inter-animation in the chat. In the text-area control, the users may enter concepts (or just click "Use concepts from assignment" link to get the default topics for the current assignment). By clicking the "SHOW" button, the inter-animation of the concepts is highlighted in the conversation graph.

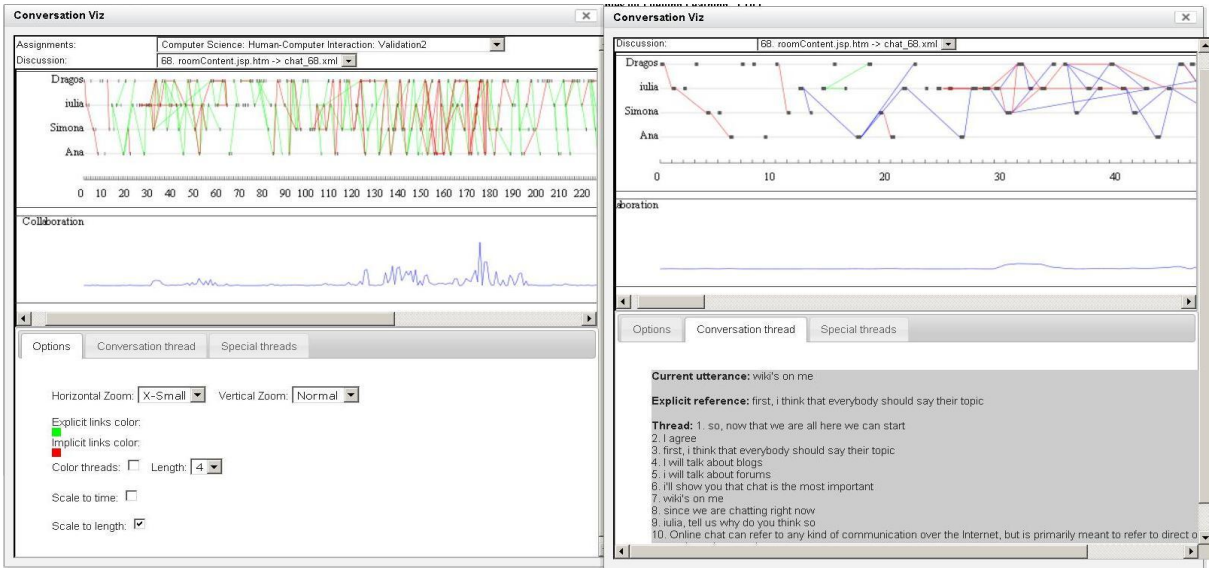


Figure 5. Conversation visualisation widget.

Participant feedback

The participant feedback widget (Figure 6) delivers feedback for each participant of the conversation and allows comparing the statistics between different participants of the same discussion. These statistics provide (among others) feedback for being on-topic (content), for being active, for having many utterances and for the social role in the conversation, etc.

Utterance feedback

This widget (Figure 7) is used to determine which utterances are considered by PolyCAFe to be the most valuable in the conversation (as represented by the score displayed next to each utterance). Moreover, the widget displays the speech acts and argumentations found automatically in each utterance. The filtering controls of the widget allow the users to see only a short summary of the chat that consists of the utterances that have received the highest grades.

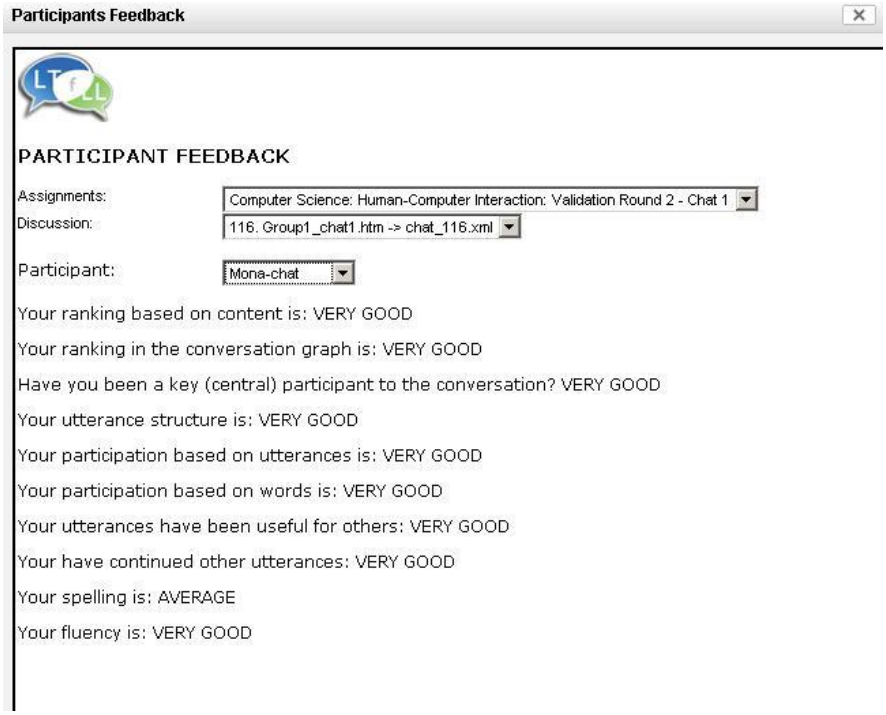
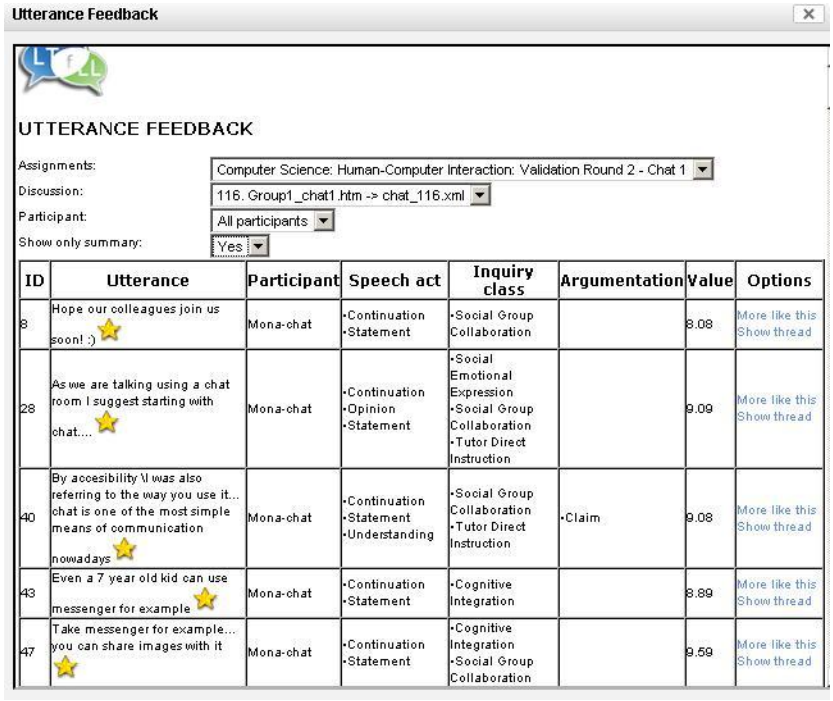


Figure 6. Participant feedback widget.



ID	Utterance	Participant	Speech act	Inquiry class	Argumentation	Value	Options
8	Hope our colleagues join us soon! :)	Mona-chat	•Continuation •Statement	•Social Group Collaboration		8.08	More like this Show thread
28	As we are talking using a chat room I suggest starting with chat....	Mona-chat	•Continuation •Opinion •Statement	•Social Emotional Expression •Social Group Collaboration •Tutor Direct Instruction		9.09	More like this Show thread
40	By accesibility \I was also referring to the way you use it... chat is one of the most simple means of communication nowadays	Mona-chat	•Continuation •Statement •Understanding	•Social Group Collaboration •Tutor Direct Instruction	•Claim	9.08	More like this Show thread
43	Even a 7 year old kid can use messenger for example	Mona-chat	•Continuation •Statement	•Cognitive Integration		8.89	More like this Show thread
47	Take messenger for example... you can share images with it	Mona-chat	•Continuation •Statement	•Cognitive Integration •Social Group Collaboration		9.59	More like this Show thread

Figure 7. Utterance feedback widget.

Enhanced search conversation

The enhanced search conversation widget (
Figure 8) offers a ranking of the participants of the conversation or of the utterances that compose it, given a search query. The ranking takes into account not only lexical scores (just like a normal Google search), but also semantic scores for the words very similar to the ones in the query and a social network score for each utterance and participant. The widget calls the Java-based search service developed for PolyCAFe.

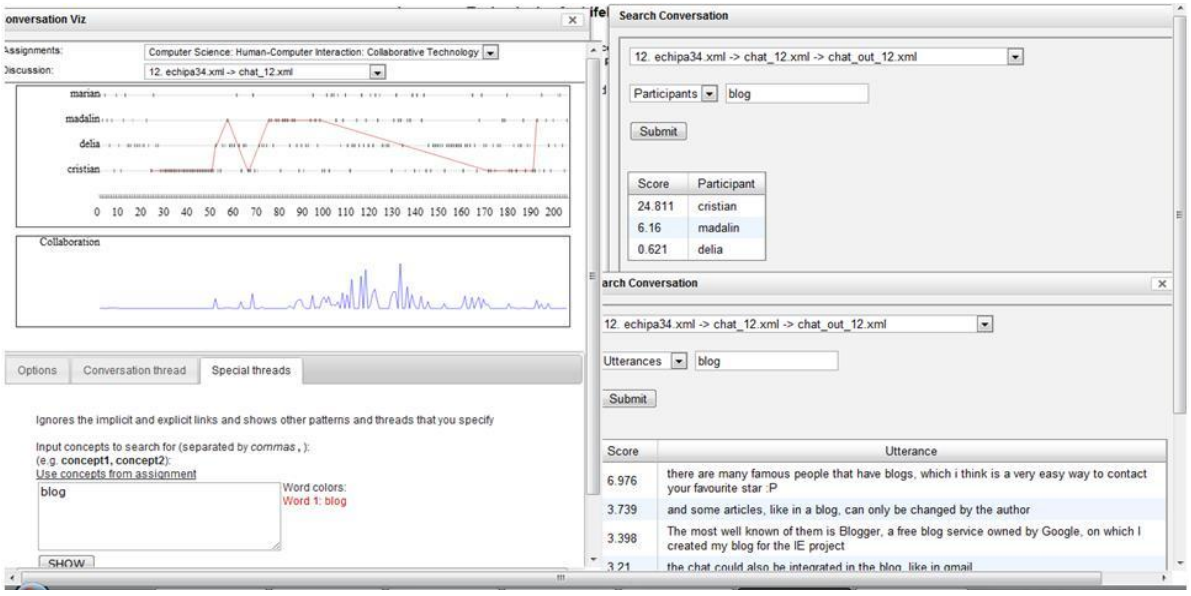


Figure 8. Enhanced search conversation widget.

Assignment management

This widget (Figure 9) permits basic operations for defining, editing and deleting assignments. It has an upper area with the controls needed for adding or editing the information that defines an assignment: course, LSA space, main concepts, chat or forum, etc. In the lower part, the list with all the previously defined assignments is presented to the users ordered by the insert time. The widget communicates only with the PHP-services using JSON.

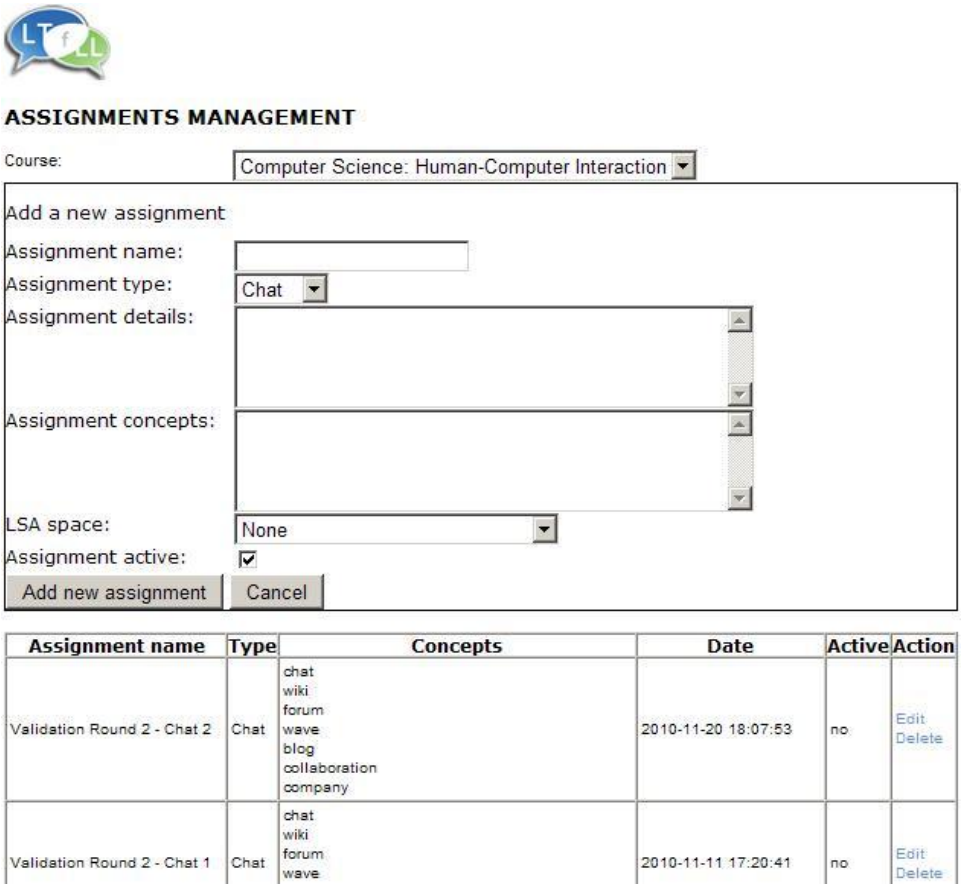


Figure 9. Assignments management widget.

Conversation management

The functionality of this widget (Figure 10) is similar to that of the "assignment management" widget; the only modification is that this widget allows CRUD operations for conversations (chats or discussion threads).

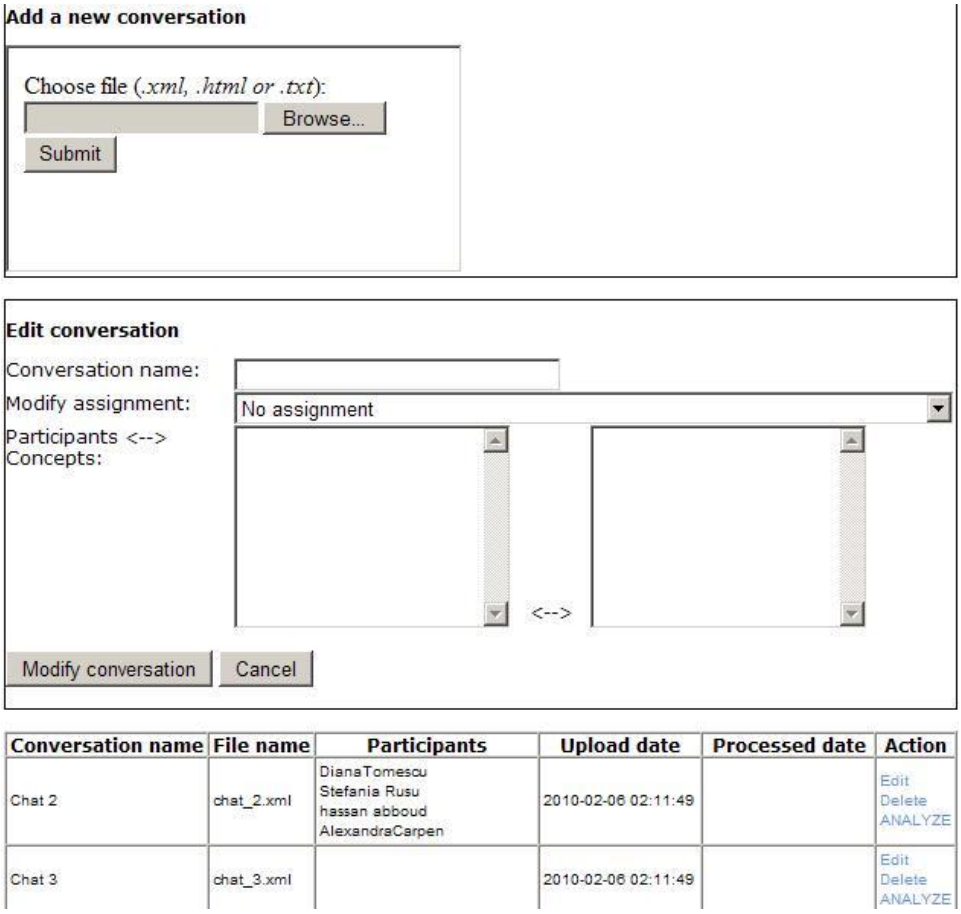


Figure 10. Conversation management widget.

3.3 PenSum (T5.2)

So far, PenSum features one widget (shown in Figure 11), which is dedicated to providing the student with synthesis writing assistance. It allows users to get feedback on their syntheses whenever they want. PenSum uses an LSA-based web service to perform sentence comparisons in order to evaluate:

- which sentences of the original course do not seem to be taken into account in the synthesis;
- what synthesis sentences seem to be off-topic;
- between which synthesis sentences there seems to be a coherence gap.

The widget also allows the learner to question the feedback of the system and justify it by explicitly linking sentences.

Versions 1 and 1.5 did not allow the learner to add any learning resources to the system, which is why T5.2 plans to use communication with iFLSS to allow the learner to import new documents, which then can be subsequently summarised. Still the difficulty of adequately extracting which parts of a webpage are relevant to the actual content of the text and which are not, make the use of this feature limited.

Since version 1.5 of the system, the learner's syntheses are stored as different versions, which are used to feed Conspect, in order to allow users to analyse their own productions.

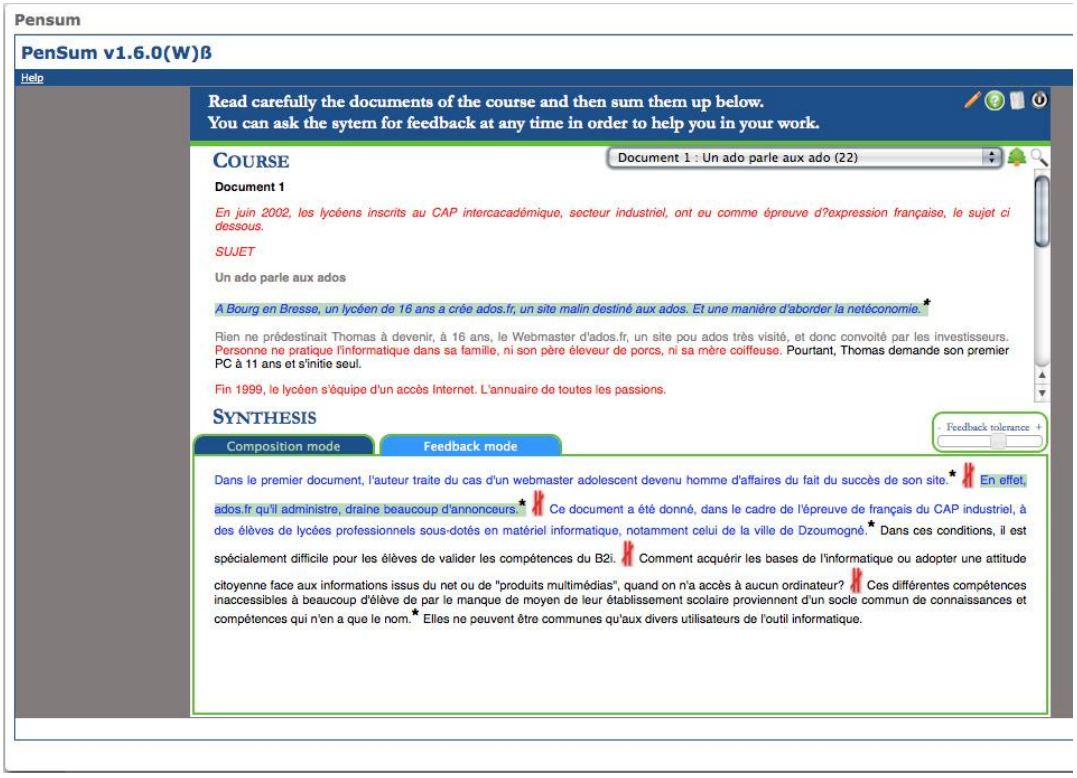


Figure 11. PenSum widget.

3.4 iFLSS (T6.2)

Definition retrieval

The widget in Figure 12 shows a definition taken from Wikipedia of a given term. The term is received from the search input or from the user's clicking on a concept in the ontology visualisation widget (see Figure 13). The widget gets the definition from Wikipedia and displays it inline. The widget takes into account the concepts related to the term given in order to disambiguate the definitions coming from Wikipedia. For example for Java there are more than ten different articles in Wikipedia but only the one related to computer science is returned if a user clicks on the Java concept in the Computer Science ontology.

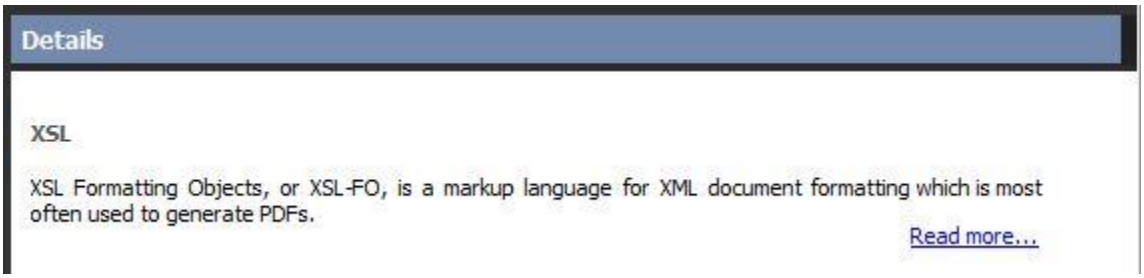


Figure 12. Definition widget.

Ontology visualisation

The widget for the ontology visualisation (Figure 13) accepts a keyword and retrieves the appropriate concept from the ontology, together with its closest related terms and displays them as a graph. The widget also emphasizes the source of the concepts through colour codes. The concepts modelled by experts are displayed in blue while the concepts discovered by social networking analysis and ontology enrichment are displayed in green. The user can click any concept in the widget and by doing this he can see the relations of the concept clicked. Also by clicking a concept, a search is triggered in the nearby widgets showing associated resources from social networks.

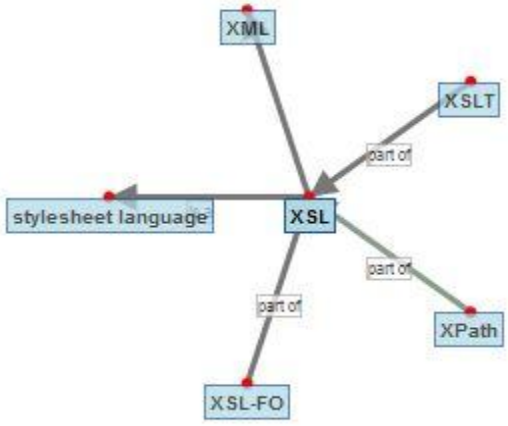


Figure 13. Ontology visualization widget.

Search results widgets (for YouTube, SlideShare, Delicious, Bibsonomy, including ontology-based filtering)

These widgets show live keyword based results from different social networking sites like Youtube.com, Slideshare.net, Delicious.com, Bibsonomy.com using specific API's offered by each of these sites (

Figure 2 and Figure 14). The results returned from these sites are filtered according to the current ontology. The filtering is done through the disambiguation of terms in the results returned. For example when searching for XSL the results returned can be referring to the XML technology or to the "XSL Slot Machine". The results are filtered on the fly and the results which the algorithm doesn't consider to be correct are displayed in faded grey. The widget receives the query terms from the search input or from the user clicking on a concept in the Ontology visualisation widget

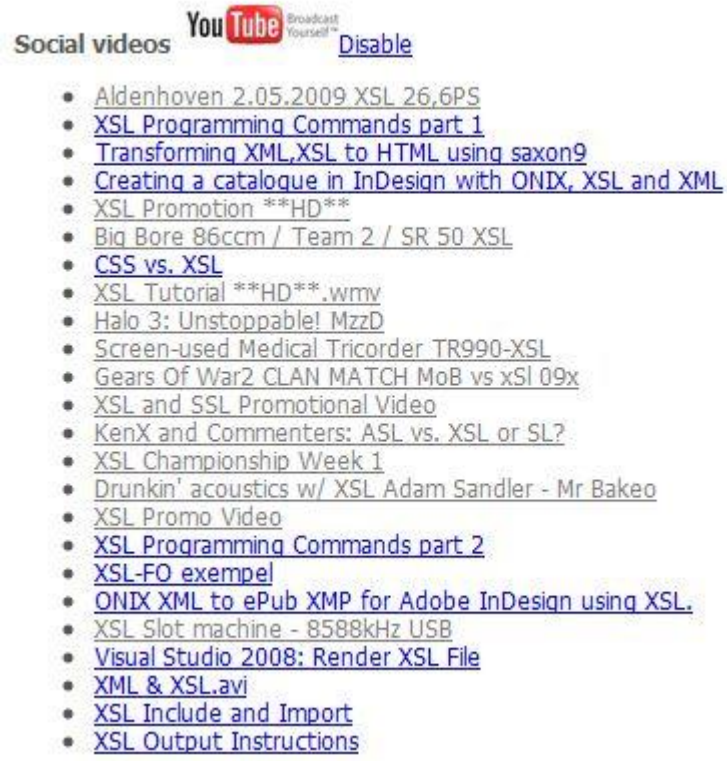


Figure 14. Social video widget.

Social search results

The widget accepts a query input and shows results coming from the social network of a given user. The user can be a tutor or a student, according to the configurations of the learning management system where the widget is embedded. Like the social people search results widget and the social personalised recommendations widget it produces valid results only if the owner has relevant resources in his social network.

The widget presents the list of resources from the social network together with the tags that have been used to describe that resource and with an icon that shows where the resource comes from (YouTube, Delicious, SlideShare, etc.). The resources are ranked by the popularity of the tags used and of the users that tagged the resource and of course according to the matching between the tags used and the query terms introduced by the user.

Social people search results

The social people search behaves just like the social search with the notable difference that it returns links to users on social platforms based on how many relevant resources those users have for the given query terms.

The social people search can be used to rapidly identify the people close by (from a social networking point of view) who are interested in a given subject. The people returned come from the social network of the owner of the widget.

Social personalised recommendations

The social personalized recommendations (Figure 15) are links suggested to the user. The links point to resources in the social network of the owner of the widget. The recommendations are computed based on affinities between the owner of the resource and the sets of tags with which the resources are tagged and based on the affinities between tags and resources. Affinity is in this case expressed as how often two items appear together in a tagging event.

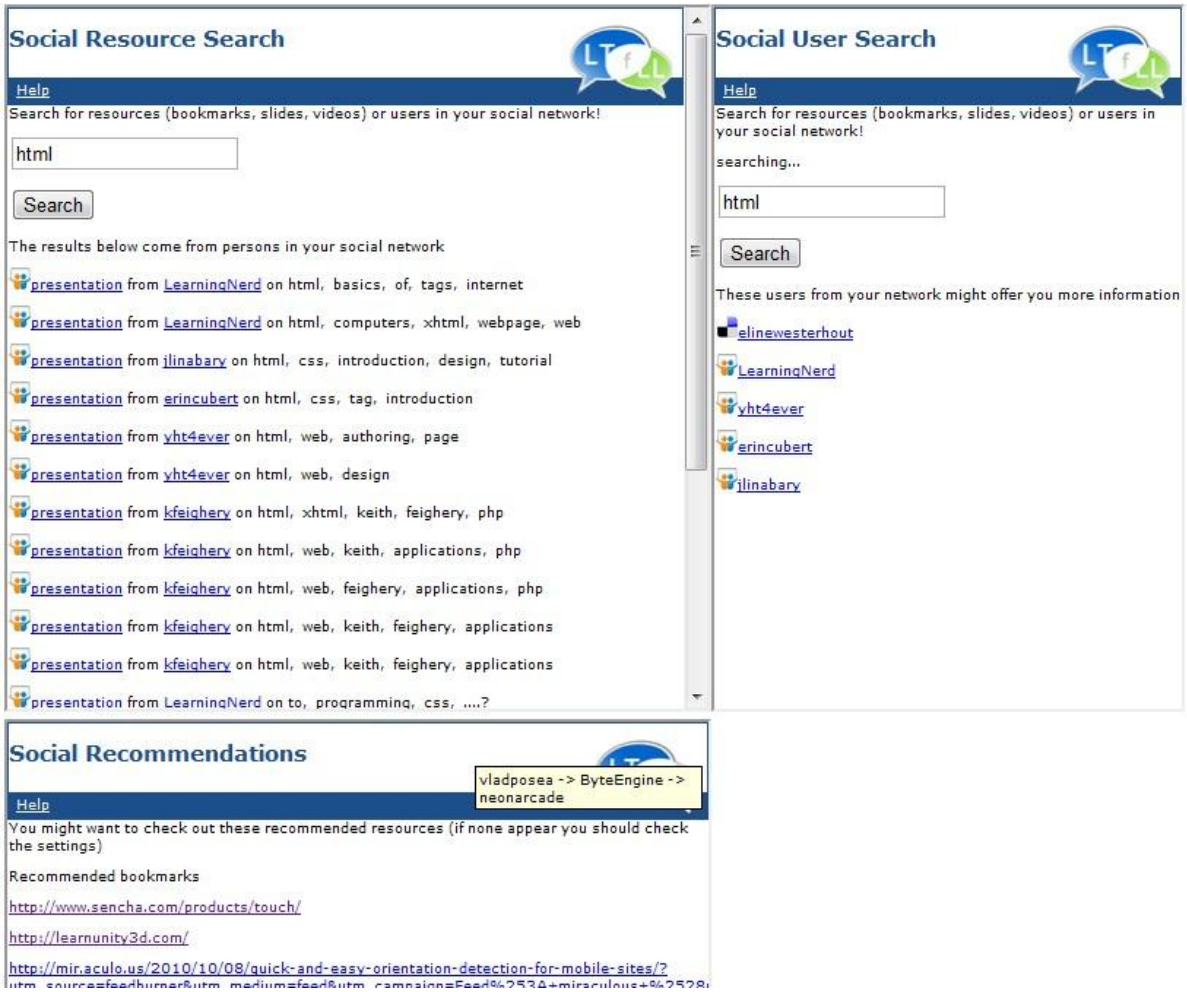


Figure 15. Social personalised recommendation widget.

4. Widgets arranged in a 'short' thread

This section describes the short thread that includes the LTfLL Positioning Service (LeaPos) and the Formal Learning Support System (FLSS). As already explained in D4.3.1, the knowledge rich conceptual feedback approach of LeaPos previously already implemented some of the FLSS functionalities. The short thread integration of LeaPos and FLSS functionalities significantly deepens the data communication and integration between both services. It facilitates the semi-automatic lexicalisation of LeaPos ontology within FLSS, while the LeaPos tutor adds missing lexicalisations that are stored by FLSS and then used by LeaPos for an improved and automated annotation of learning materials and answers. Together with the new ability to annotate questions with the concepts discovered in learning materials, this new functionality enables LeaPos tutors to build and tune formative conceptual feedback via an intuitive interface. FLSS language experts can then decide which new lexicalisations suggested by the LeaPos tutors can be incorporated into the ontology.

Additionally the Wookie widget integration of the LeaPos student view is documented, greatly reducing installation costs and facilitating further possible workflow integration with other tools of the LTfLL project.

4.1 LeaPos (T4.1)

LeaPos supports tutors and provides formative feedback to learners. It was documented in D4.3.1. Figure 16 shows the output of the LeaPos widget analysing conceptual coverage of learner answers. When the learner requests this kind of feedback for his answer, the widget presents a table to the learner with concepts divided into three columns: "Found", "Missing" and "Additional" concepts. For more details on this you might want to refer to D4.3.1.

The screenshot reveals a new feature that pops up when clicking either of those concepts, showcasing part of the deeper integration between LeaPos and FLSS: The widget also displays the concept details, helping the user understand the output better:

- the concept class URI
- the concept description
- the concept lexicalisations
- relevant text snippets of learning materials containing the concepts

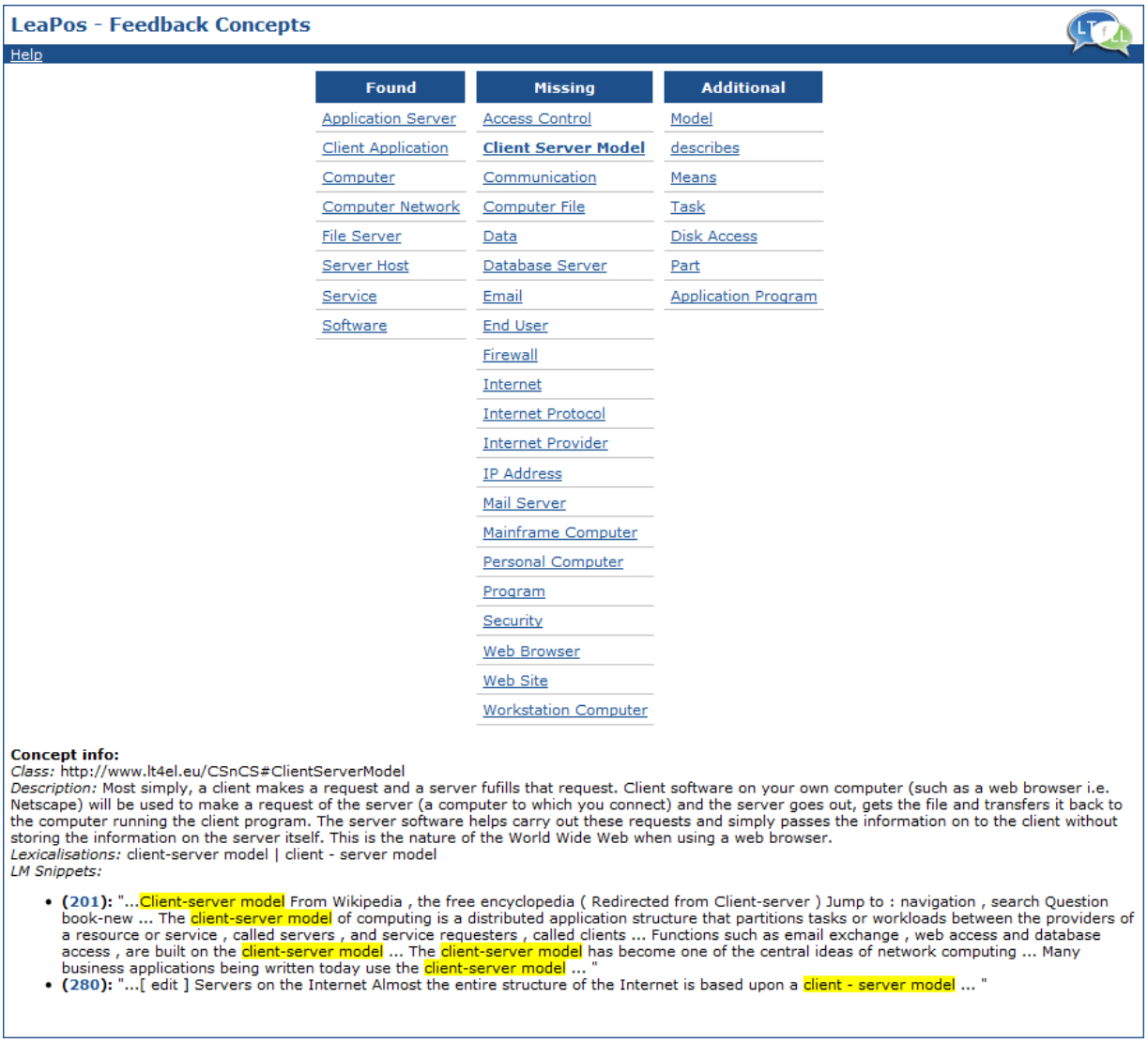


Figure 16. Feedback widget.

The concept description and the concept lexicalisations are accessed by calling remote web services from FLSS. However, the main aspect of the integration was a real workflow integration, allowing less experienced tutors to intuitively annotate questions themselves and thus paving the way for the conceptual feedback widget to work for new questions also, instead of only those hard-coded by experts on the FLSS side. For further details on the theory behind the integration process please consult D4.3.1.

The following paragraphs showcase the technical and user interface integration on the LeaPos side and how questions can be prepared to work with the conceptual feedback widget. For this purpose the tutor has to use the standalone version of the LeaPos tutor interface, since as of now it has not been widgetised.

In order to provide knowledge rich feedback outlining the conceptual coverage, the answers need to be annotated. For that purpose the remote FLSS annotation web service is called passing along the learner's answer and returning it in annotated XML. After that, the XML is parsed for the concepts to compare them with the concepts required for the question.

For the tutor to be able to annotate the questions, a new feature had to be implemented as shown in Figure 17. There the tutor is able to assign an existing concept from a learning resource relevant to the question instead of manually entering the concept classes. This was done for several reasons: First it is a lot easier for a tutor to select the right concepts if they are already filtered (in this case with the help the learning materials) and not having to find the concepts in potentially huge and numerous ontologies. This also leads to the possibility of adding learning material references, where the learner can easily find the relevant passages in the recommended learning materials (Figure 18). Secondly this helps the tutor to ensure all the required content from the learner is actually covered by the learning material.

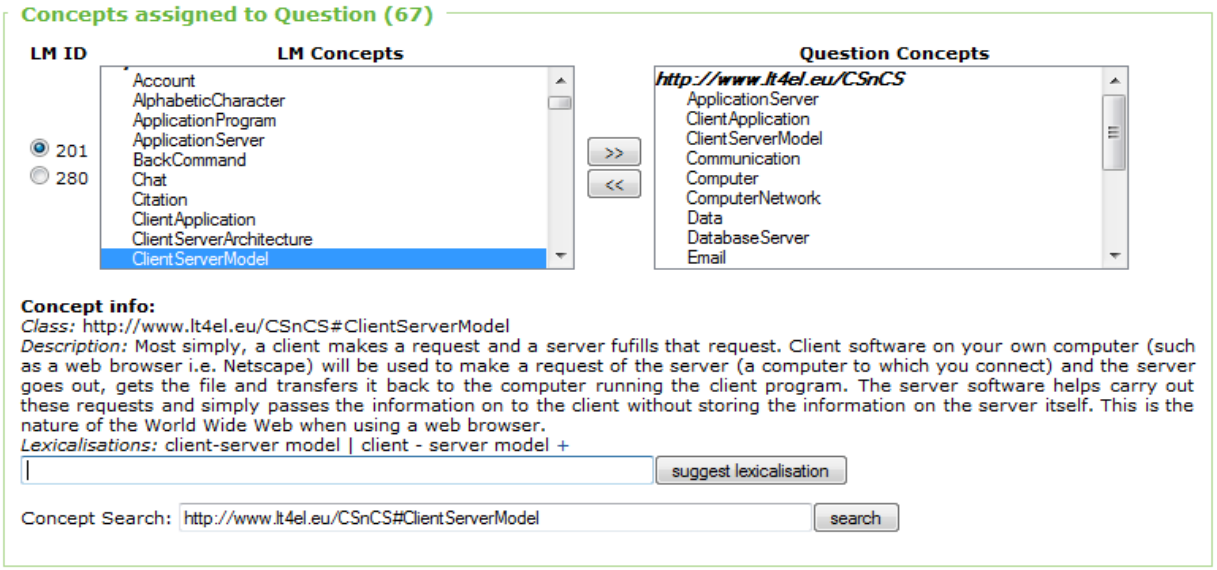


Figure 17. Concept widget.

The actual module as depicted in Figure 17 allows the tutor to select the relevant learning materials via the radio buttons in the "LM ID" column on the left. The column "LM Concepts" to the right lists all the concepts identified in the learning material. With the arrow buttons in the centre the tutor can finally assign or remove the selected concepts to the question. The concepts then listed in the "Question Concepts" column are the ones required by the knowledge rich conceptual feedback widget.

Every time the tutor clicks on a concept its details are shown below in a similar fashion as in the feedback widget explained earlier, making use again of the FLSS web services. There are, however, two additional input boxes: At the bottom there is the "Concept Search" input box, which lets the tutor search for concepts in the relevant learning materials, filtering the "LM ID" column appropriately and automatically selecting the correct concept class in the "LM Concepts"

list. The second input box may be utilized to suggest additional lexicalisations to the FLSS annotation service. Suggestions can be remotely stored, viewed and deleted, until a language expert on the FLSS side approves or rejects the suggested lexicalisations and integrates them within the regular annotation service and ontology. The web service API for the FLSS suggestion system was laid out in D4.3.1.

To bridge the time gap for this approval process, the answers and learning materials actually are annotated twice: First the content is run through the FLSS annotator web service normally, returning the already accepted concepts contained in the answer or learning material. In a second iteration after that, the current lexicalisation suggestions are queried from the FLSS web service and used to test the content for the additional conceptual coverage.

The above implied that not only answers are annotated (on-the-fly), but also the learning materials. The learning materials are annotated every time they are uploaded, or their textual content has been cleaned up with the content edit form in the "Inspect Learning Material" module first introduced in D4.3.1. In the screenshot displayed in Figure 19, the concept details field has been added as well, activating whenever the tutor selects a concept contained in the learning material.

When clicking the "edit" button right below the listed annotations, the tutor is redirected to use FLSS to manually edit the already automatically annotated concepts and add new lexicalisation suggestions. The lexicalisations generated during the refinement process are stored by FLSS and can be then used by experts to update the ontology lexical database. This is an alternative way to add lexicalisations to the integrated system, using the same web services in the back and having the exact same impact as the procedure outlined above and in Figure 17. The latter is more convenient, requiring fewer steps to add lexicalisations, while using the annotation editor (described in subsection 0) is visually more appealing and offers more options to the experienced user.

During the preparation of the knowledge rich feedback and the assigning of concepts to the various questions, heavy use was made of the lexicalisation suggestion services, especially for the German course, where the initial number of lexicalisations was very sparse.

In another note, LeaPos has been fully widgetised to allow for a greatly simplified installation process in any Wookie widget enabled environment as well as creating new potential for workflow integration with other widgets. As of today only the student view has been widgetised, while the more complicated tutor view will be widgetised later on. Figure 19 shows a Wookie widget configuration with all four LeaPos student widgets enabled: the student course management view plus the three different feedback widgets.

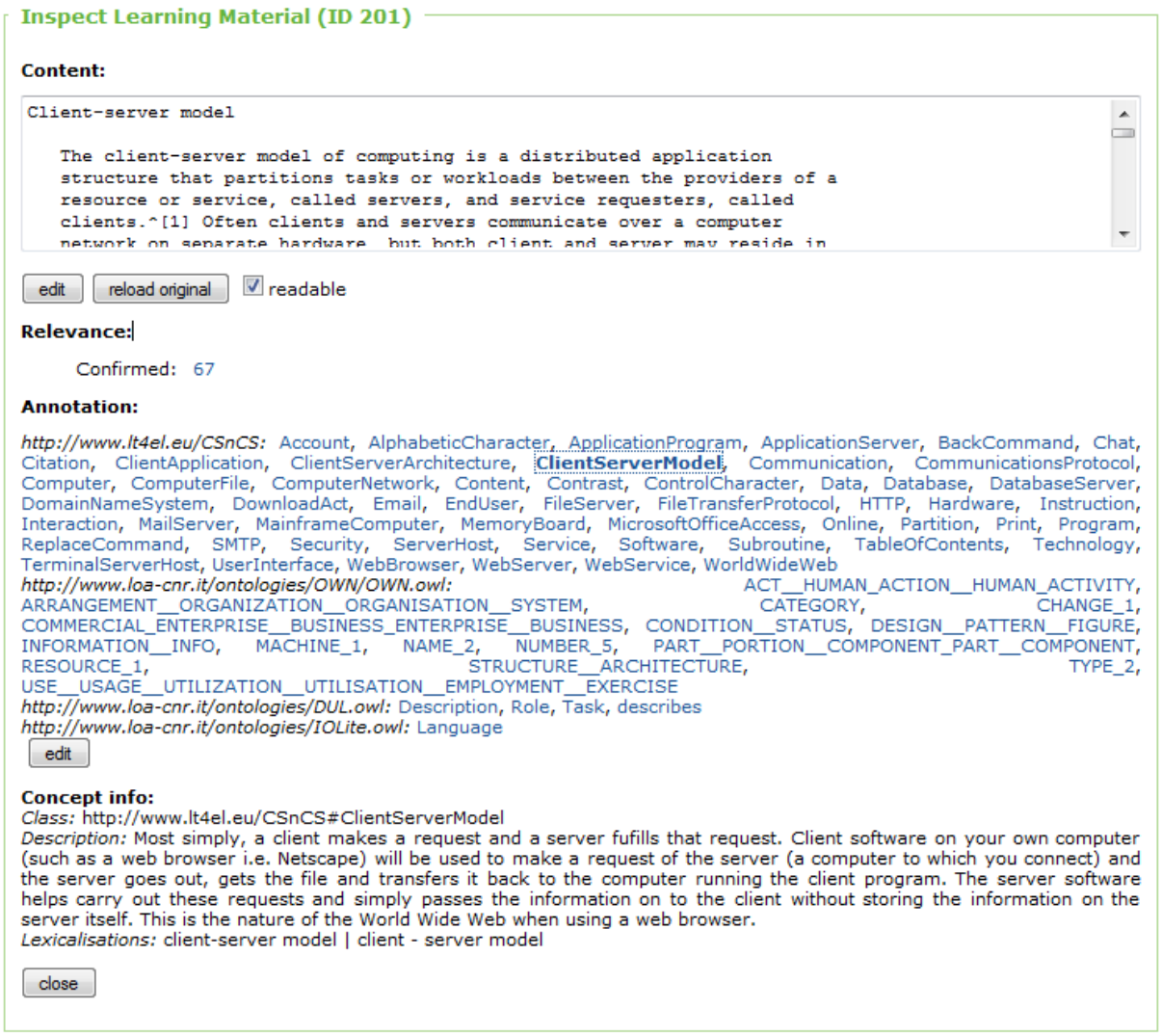


Figure 18. Learning Material widget.

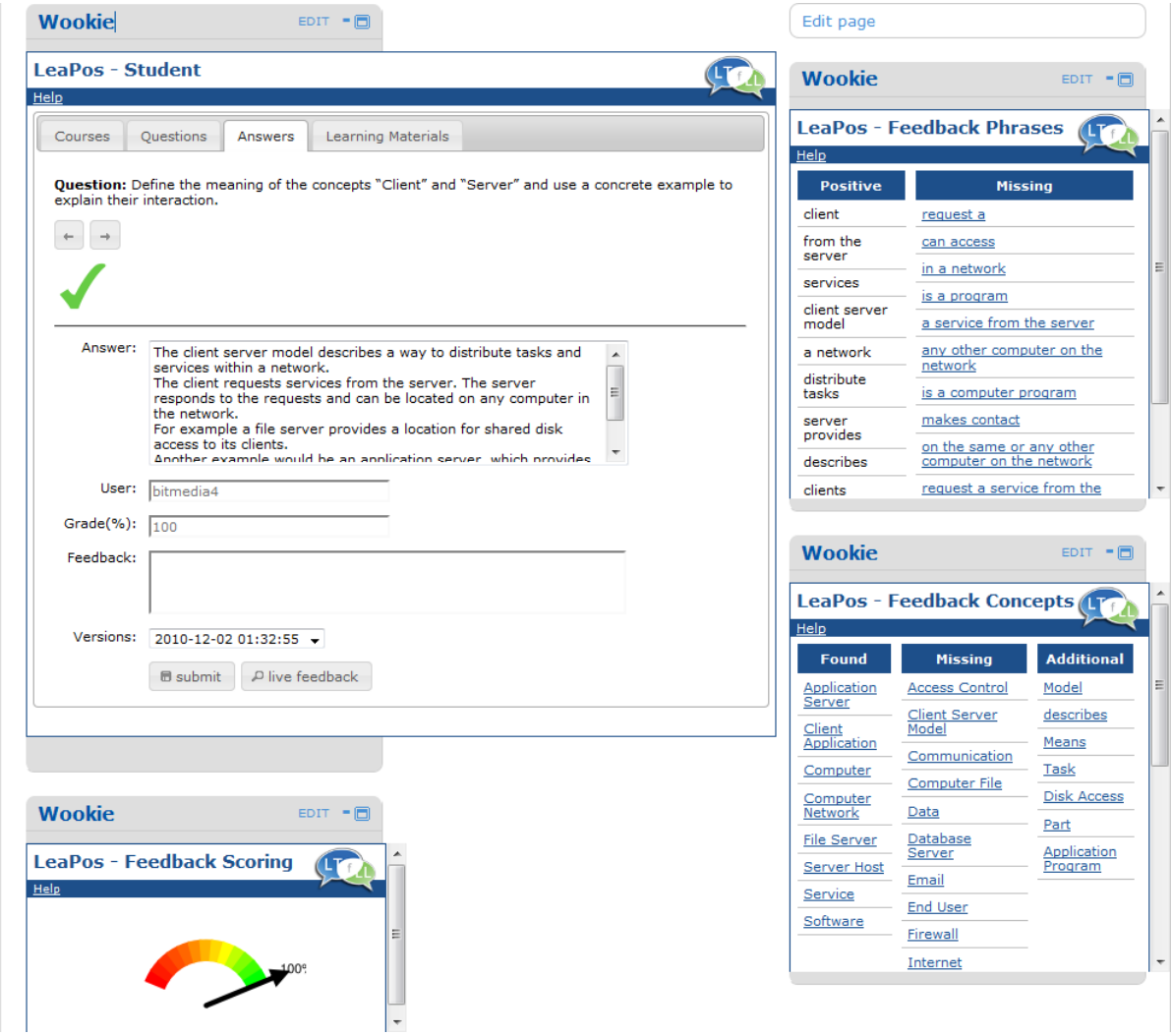


Figure 19. Short thread widget arrangement shown in Elgg.

4.2 FLSS (T6.1)

The primary purpose of the tool is to support management of semantically annotated textual resources. This includes resources visualisation, editing, creating new resources, publishing at and searching in the semantic repository.

The visual part of the component comes in two flavours - as a standalone web application and as an integrated Wookie widget. In either way the supported functionalities are identical and it is up to the user to decide which one to choose (the former one offers more screen space).

From an architectural point of view, the component comprises a visual front-end for the users and a set of web services supporting different annotation tasks. Each of the services can be used

independently by any internal or external component or tool via uniform communication protocol (i.e. HTTP). Among the more significant ones are:

- annotation service - performs automated semantic annotation of textual resources. Input is plain text in English, German or Bulgarian and the output is an XML document containing the analyzed and annotated text. API specification is available at SF (see references).
- lexicalisation service - supports bidirectional mappings between ontology classes and their corresponding lexicalisations (in English, German or Bulgarian). Additionally the service delivers textual descriptions of the various ontology concepts. API specification is available at SF.
- semantic search - performs document searching in the semantic repository based on the semantic annotations. The input queries might contain keywords (in English/German/Bulgarian) and/or ontology concepts (URIs). For more advanced usage, the service exposes SPARQL endpoint. API specification is available at SF.

The visual interface reflects the integrated-in-the-background services in an IDE-style environment, providing separate views for searching, ontology and document annotation visualisation (Figure 20).

For improving reusability and integration, the web GUI component itself can be integrated with other web components that must provide 'source' and 'update' parameters for input and result update locations. This way, third party software components can trigger visualisation/editing of annotated textual resources on demand.

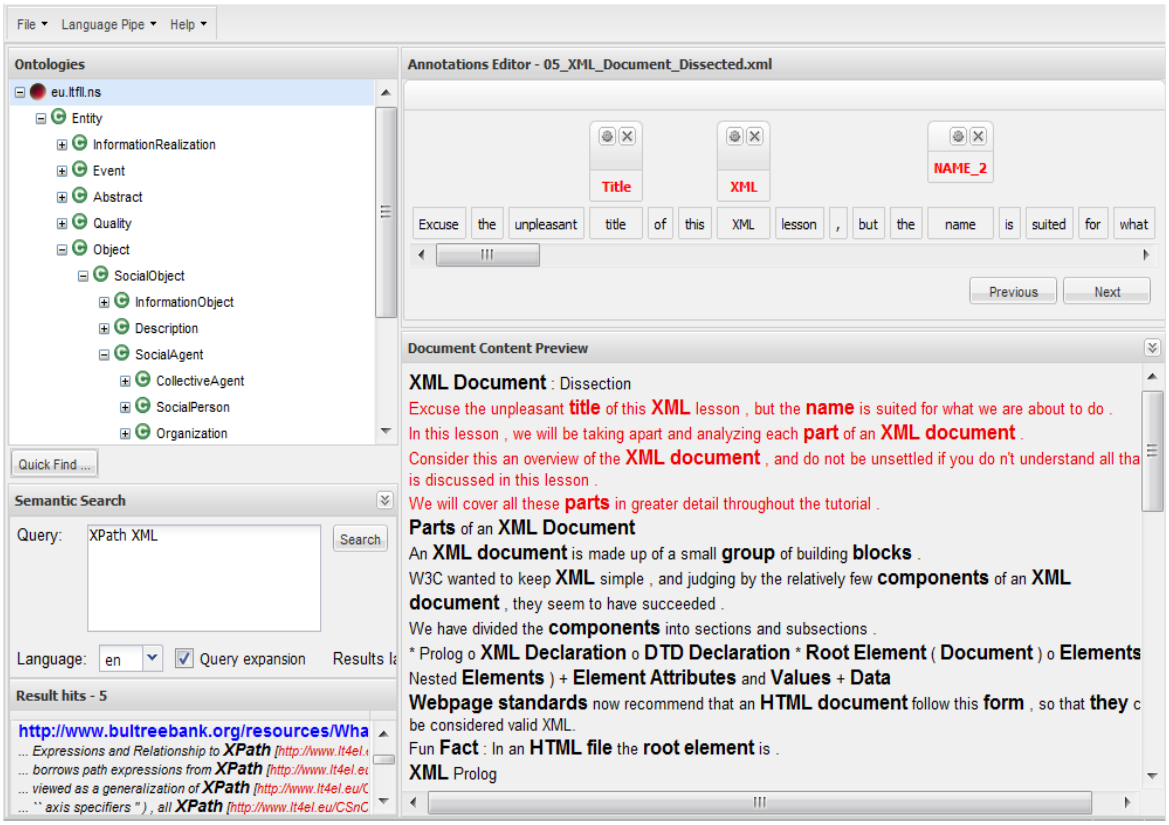


Figure 20. Web-based user interface of FLSS.

5. Interwidget communication components

The following subsections describe technical improvements regarding Interwidget communication (IWC, see also Hoisl et al., 2010a) to make integration of the long and short thread possible. Therefore, we give an update of work done related to the Wookie engine, the Elgg plug-in, the widget template, and the OpenACS widget server and plug-in prototype.

5.1 Wookie widget server

Our previous improvements to Wookie had the disadvantage that we created a branch specifically targeted to our purposes and not in-line with the development trunk (see D2.3, pp. 36). Therefore, we could not benefit from new improvements, such as new functionalities or bug fixes, committed from the community. We eliminated this duplication by integrating IWC specific functionalities in the head revision of the Wookie development version once again taking into account our former implementation and improving it significantly. The outcome is a more generic approach of data sharing strategies with which we are now back in line with Wookie's newest development version.

By learning from our first approach, we could simplify our code while making it target a wider range of scenarios. Our main application - aiming at notifying all widgets in one user space (see D2.3, p. 37) - hasn't changed, but other strategies can be applied as well. Therefore, a Wookie administrator is now able to couple widgets by defining matching parameters. All widgets having these variables in common are treated as belonging together, therefore share data and are notified of updates.

In file widgetserver.properties one has to state all coupling IWC variables. Currently implemented are the following parameters: sharedDataKey, apiKey, idKey, userId. With a few lines of code it is no problem to add new coupling parameters to the existing ones. Any combination can be used for binding widgets together:

- sharedDataKey: The key generated by an application representing a specific context (e.g. a page, post, section, group, user or other identified context).
- apiKey: The key issued to a particular application.
- idKey: An identifier representing a single widget instance.
- userId: An identifier (typically a hash rather than a real user Id) issued by an application representing the current viewer of the widget instance (Wilson, 2010).

Our purpose fits a coupling of apiKey with userId. Therefore, we restrict data sharing to widgets of one user (userId) in one specific application (apiKey). The apiKey is specific to our Elgg installation (cf., e.g., the Integration Report, p. 23) and the userId represents a user's unique ID in Elgg (as defined through our Elgg plugin, Hoisl, 2010a). Hence, all widgets belonging to a logged-in user in Elgg hare data and receive notifications. Another broader linkage strategy could be just using the apiKey and letting all widgets of one application share their data. If someone would like to completely avoid sharing data, the coupling parameter has to

include the IdKey. As it identifies a single widget instance no other widget will match this criterion.

The workflow of our IWC implementation is to firstly find out all widget instances matching the coupling statements. Then, shared data is duplicated and distributed to all instances found combined with a notification about the updates. Therefore, we only had to modify and extend setter methods of data sharing functionalities together with notification behaviours, but could leave data fetching procedures of widget instances basically as they are.

We provided a patch for the newest Wookie revision and also four test widgets using our newly development IWC functionalities (see Wilson, et al., 2010). The first two test widgets simply store and retrieve strings inserted by a user and display them. The other two widgets use not only our LTfLL widget template with its design but also our IWC widget library for storing and fetching shared data using JSON (described in detail in Section 4.3). Therefore, our developments are getting tested not only within our project, but also by a wider community of interested people. The patch with its test cases is currently under review and we expect that it is going to be integrated in the development version of Wookie soon. Unfortunately, there have been some delays as a first official release of Wookie is planned at the moment and integration of new functionalities is generally postponed after the release. In the meantime, our modified Wookie Server capable of IWC can be obtained from LTfLL's SourceForge repository.

5.2 Elgg Plug-in

A new version 2.2 of our Elgg plugin for Wookie widgets was released. New functionalities include:

- The URL to the Wookie engine and the requested API key can be set through the tool administration menu in Elgg - there is no need to edit files anymore.
- Rudimentary OpenID support is available (in combination with the OpenID client plugin for Elgg (Jardine, 2010)) - the OpenID identifier gets passed to a Wookie widget via the URL parameter openid_identifier.
- Uncoupling widgets from the dashboard and enlarging them (even to full screen) is possible through the integration of Highslide JS (Honsi, 2010). When hovering the mouse over a widget, a small icon appears in the top left corner. Clicking on it uncouples the widget from the dashboard and displays it in an overlay window. This can be done with an unlimited number of widgets. The uncoupled widget is resizable and draggable. Navigation between widgets in a dashboard can be done through the next and previous arrows or by using left and right keys.
- Some minor bug fixes.

Furthermore, we are working on a release of version 2.3 currently deployed at our development server. A pre-release version is available at our SourceForge development repository. It will cover the following new or improved functionalities:

- The icon for uncoupling widgets from the dashboard is newly positioned. As the icon could overlap with content of the widget it is now integrated in the general header of dashboard widgets, next to the edit and minimize buttons.
- A special parameter is set (expand=true) when uncoupling widgets from the dashboard to differentiate them from dashboard widgets.
- The list of selectable widgets is sorted alphabetically now.
- Improvements to error messages were made.
- Code restructuring took place.
- Some minor bug fixes (e.g. to prevent browsers to cache widgets).

5.3 Widget template

WP2 has developed and distributed a widget template to be used by every partner for version 1.5 of the services. This was done in order to homogenize the design ('look and feel') and also to provide a couple of helpful functionalities used by every developing partner (e.g. a set of IWC methods). Following is a brief explanation of how to configure LTfLL widgets and make them work with IWC.

Explanation of directory structure:

- ./help/
  Help pages should go in here.
- ./images/
  Put images in this directory.
- ./legal/
  Put any legal texts as well as copyright statements here.
- ./lib/
  Any general JavaScript libraries are inserted here.
- ./ltfll-design/
  LTfLL design specific CSS, JavaScript functions, and images reside here.
- ./scripts/
  Any widget specific JavaScript functions are put here.
- ./style/
  Widget specific CSS are located here.

In order to customize a widget, some files have to be edited:

- ./config.xml
  The main file describing a widget (e.g., name, author(s), widget ID etc.). The polling feature needs to be removed if the widget does not need to listen for IWC updates because otherwise it will just generate useless traffic.
- ./index.html
  The HTML start page. Need to be configured accordingly (e.g. HTML widget title and of course the actual content the widget should display).
- ./strings.js
  The widget's title to be displayed as a headline in the LTfLL specific design has to be inserted here together with the name of the help link.
- ./scripts/index.js
  It can be configured if the LTfLL logo in the upper right corner should be shown or not. Furthermore, it has to be indicated if the help link and the footer (and therefore status messages) have to be displayed. This is done by setting the following three variables to either true or false: showLogo, showHelp, showFooter.
- ./help/index.html
  The start page of help sites associated with this widget.

Basically, for setting up IWC there are two data format options: JSON or key/value pairs. It is recommended to use JSON, but for some circumstances key/val pairs might be better suited. One have to bear in mind, that all IWC specific JSON methods end with 'JSON' (e.g. IWCsetVarJSON()). Mixing up JSON and non-JSON methods will result in data loss and erroneous behaviour.

A widget can use IWC by configuring the following:

- ./config.xml
  If a widget wants to receive updates on shared data, one has to be sure to include the polling feature statement. If a widget just wants to set shared data, but does not want to get notified about updates happening to data, then the line needs to be removed because otherwise it will just generate useless traffic.
- ./scripts/index.js
  By calling IWCinit() one has to specify if JSON or non-JSON (key/value pairs) should be used as data format and if already existent shared data should be initialized at widget onload. Recommendation for all widgets: IWCinit('JSON', true).
- ./scripts/config.js
  It can be configured if IWC related status messages should be displayed in the footer of the widget by setting IWCstatusMessages to either true or false. Furthermore, dependent on the IWC data format choice the following has to be specified:
  - JSON
    Variable IWCnameSpace has to be set to the namespace a widget belongs to (it is recommended using work package numbers, e.g. 'WP6').

    - non-JSON
      All IWC variables a widget should listen to for updates must be stated in array IWCsharedData.
- ./scripts/iwc-callbacks.js
  All call-back functions which are invoked when fetching IWC updates have to be defined here. They have to be named exactly as the IWC variables a widget listens to, but with 'IWC' in the beginning (e.g. IWC variable: keyword -> IWC call-back function: IWCkeyword()).

Until now it has been described how IWC variable updates get retrieved and a widget gets notified. There are two further functionalities: setting and deleting shared data. Again it is distinguished between JSON and non-JSON:

- JSON
  - IWCsetVarJSON(ns, name, value)
    This function sets a new attribute with name and value in the defined namespace (ns).
  - IWCdelVarJSON(name)
    Deletes the attribute labelled name (this can only be done for the namespace the widget belongs to).
- non-JSON
  - IWCsetVar(name, data)
    This method sets a new variable name with value data. As here IWC variables do not belong to a namespace, any can be set.
  - IWCdelVar(name)
    Deletes variable called name. As here IWC variables do not belong to a namespace, any can be deleted.

All fetched IWC variables are locally stored in array IWCdata[name] and can be addressed accordingly. Furthermore, with function setStatus(status) (and setStatusFade(status) respectively) IWC specific status messages are set and displayed in the footer of a widget (only if both showFooter in ./scripts/index.js and IWCstatusMessages in ./scripts/config.js are set true). If one wants to set own status messages these functions should be used, as well (but one has to be sure that showFooter in ./scripts/index.js is set true, otherwise no footer and therefore no status messages will be displayed). For deleting status messages function delStatus() has to be called. If the status message should automatically fade after a couple of seconds function setStatusFade(status) should be used.

If a widget has been configured as described above, it will receive updates on IWC variables it listens to. If by any means a widget needs to retrieve an IWC variable which it does not listen to (because the polling feature might be disabled), it can be done as follows:

- JSON
  All IWC variables for the namespace a widget belongs to are retrieved automatically

(because it is one JSON object) and they can be addressed using IWCdata[name]. If polling is disabled and all IWC variables for a namespace should be retrieved, IWCgetVarJSON() has to be called. This makes no sense when polling is enabled, because a widget with polling enabled listens on all updates happening in its namespace, therefore stored values in IWCdata[name] are up-to-date anytime.

- non-JSON
  IWCgetVar(name) has to be used to fetch an IWC variable not listened to. Once retrieved, the variable is also stored in array IWCdata[name]. If an IWC call-back function has been defined for the variable it will be invoked, as well.

With IWC shared data is set and it is listened to shared data updates. If there is an update for a variable a widget listens to, the corresponding IWC call-back function is invoked doing something with the newly received data. For the JSON method it is enough that the call-back functions are named as described above. For the non-JSON method all IWC variables a widget wants to listen to have to additionally be defined (in ./scripts.config.js, as described before). This is just to minimize traffic for the non-JSON method because it is useless to receive updates for IWC variables with which a widget does not want to do anything. Every update on an IWC variable triggers a new request, therefore it is wise to only fetch the ones needed. By using JSON all attributes set in a namespace (e.g. 'WP6') are retrieved and the ones having a call-back function are invoked. Network traffic is no problem here because a widget listens to only one namespace which is represented as once IWC variable consisting of a JSON object.

Using non-JSON, variable names have to be unique. Therefore, it is recommended using namespaces, e.g. 'WP6::keyword'. All variables in the whole scope of IWC can be set, retrieved, and deleted. Therefore, extra caution is advisable.

With JSON, variables belong to a namespace and therefore can be uniquely identified. A widget is associated with exactly one namespace and listens on updates only on that namespace. This means shared data can only be retrieved (and deleted) for the namespace the widget belongs to. The only exception of interfering with another namespace is the setting of shared data. If a widget of one namespace wants to communicate with another widget of a second namespace, it can set a new IWC variable in the scope of the second namespace. Therefore, the widget in the second namespace gets notified. If for example widget 'A::1' (i.e., widget '1' in namespace 'A') wants to talk to widget 'B::1', widget 'A::1' is allowed to set a new variable (e.g. 'B::searchterm'). So, if widget 'B::1' is listening on updates for this variable it gets notified and can do something with the newly received data. This behaviour acts as a data protection mechanism that one widget does not interfere with another unless the two have negotiated how they want to do it.

Moreover, by using the template a widget recognizes if it is served by a Wookie server or not. If the widget is deployed outside of the Wookie container or runs as a stand-alone version, Wookie specific dummy objects are created and IWC methods are overloaded. Therefore, widgets can be used elsewhere without any code change. For example, a widget can be served by the OpenACS widget server described in Section 4.4. Wookie specific functionality is then disabled by default. But it is no problem to adapt the widget, for instance, to a potentially different API of another widget server.

For further information on widget development and integration, see the widget development guide (D2.3, pp. 21), the guidelines on localization and corporate design (D2.3, pp. 33), the guidelines on IWC (D2.3, pp. 36), and also the guide to OpenID authentication (if needed; D2.3, pp. 42). A description of exposed web-services from all service prototypes can be found in D2.3, pp. 50. Furthermore, if someone needs to recall architectural design decisions of our service infrastructure, refer to D2.1, pp. 15. In addition, our architecture of creating PLEs by using widgets is explained in LTfLL's Integration Report, pp. 20.

5.4 OpenACS Widget Server

To test interoperability of services and especially widgets we have developed a prototype Widget Server being able to partially replace Wookie. Therefore, core functionalities of Wookie were implemented in the Open Architecture Community System (OpenACS#) using Extended Object Tcl (XOTcl#) as a scripting language. OpenACS is an open-source web application framework running on AOLserver# which we use with PostgreSQL# as its database. We have chosen this setting because WUW is the main developer of XOTcl and has in-depth knowledge in building and maintaining OpenACS based applications from a number of previous and ongoing e-learning projects. The OpenACS package (named 'xotcl-widgets') implementing all functionality can be obtained from Demetriou et al, 2010.

Our OpenACS implementation features both a server able to deliver widgets and a connector framework serving as a plugin for OpenACS itself. The implementation is based on these three packages: 'xotcl-core', 'xowiki', and 'xowf'. Furthermore, Wookie was used as an example and we tried to conform to its implementation (especially the REST API) as much as possible. However, just a prototype was developed integrating only core functionalities of Wookie. A widget upload workflow was defined for being able to deploy widgets on the server. Once available all existing widgets can be viewed in a gallery type of style. An XML representation is provided as used, for example, by plugins to select a widget for instantiation. Therefore, a list of widgets can be generated which can be displayed, for instance, with our Elgg plugin in the same way as with Wookie (REST API calls are identical). Instantiation of a specific widget is also done with the same API calls as with Wookie, therefore our Elgg plugin can be used without any changes. Basic support for IWC is provided by handler functions communicating with parts of the Direct Web Remoting library (DWR#), re-implementation of internal Wookie specific methods (e.g. setSharedDataForKey() etc.), and interface implementations (e.g. Google Wave API).

Furthermore, the package defines also some test cases for the connector framework and IWC. Beside tests for the integration and display of widgets in OpenACS, data sharing can be tested according to the same shared_data_key (specified at widget instantiation) or the same wiki_page_id (specified by OpenACS) or combinations of both.

With this implementation we are able to disseminate not only in Wookie and Java but also in OpenACS and XOTcl communities. Interest in our approach was recently aroused by directly started large-scale FP7 EU project iTEC# (Innovative Technologies for an Engaging Classroom) with which code was shared.

6. Conclusion

This deliverable documents the status of the PLE prototype investigated and developed within LTfLL. This includes an interwidget communication (IWC) component, a new release of the Wookie Elgg plug-in, and documentation on how it is implemented in the widgets participating in the arrangements for the short and long threads.

In addition, it provides technical guidelines and information for widget developers. It remains for WP7 to validate the pedagogical utility of the threads.

References

Demetriou, Neophytos; Hoisl, Bernhard (2010): OpenACS based XOTcl Widget Server and Connector Framework. Available at http://ltfll.svn.sourceforge.net/viewvc/ltfll/Wp2/xotcl-widgets/, last access: 2010-11-26.

Hartmann, B.; Doorley, S.; Klemmer, S. (2008): Hacking, Mashing, Gluing: Understanding Opportunistic Design, In: Pervasive Computing, July-September.

Hoisl, Bernhard (2010a): The Elgg Community: Wookie Widgets 2.2. Available at: http://community.Elgg.org/pg/plugins/hoisl/read/385029?release=420305, last access: 2010-11-25.

Hoisl, Bernhard (2010b): Wookie IWC Server. Available at: http://ltfll.svn.sourceforge.net/viewvc/ltfll/Wp2/Wookie-iwc/, last access: 2010-11-25.

Hoisl, Bernhard (2010c): Elgg Plugin for Wookie Widgets - Development Version. Available at: http://ltfll.svn.sourceforge.net/viewvc/ltfll/Wp2/Wookie-Elgg-plugin/, last access: 2010-11-25.

Hoisl, Bernhard; Drachsler, Hendrik; Waglechner, Christoph (2010): User-tailored Inter-Widget Communication - Extending the Shared Data Interface for the Apache Wookie Engine. In Proceedings of the 13th International Conference on Interactive Computer Aided Learning (ICL 2010), pp. 1123-1131, Kassel University Press.

Hoisl, Bernhard; Essl, Markus; Kometter, Helmut (2010): LTfLL Widget Template. Available at: http://ltfll.svn.sourceforge.net/viewvc/ltfll/Wp2/widgets/_template/src/, last access: 2010-11-25.

Hønsi, Torstein (2010): Highslide JS - JavaScript Thumbnail Viewer. Available at: http://www.highslide.com/, last access: 2010-11-25.

Jardine, Kevin (2010): The Elgg Community: OpenID Client. Available at: http://community.Elgg.org/pg/plugins/kevin/read/433999/openid-client, last access: 2010-11-25.

Palmer, Matthias; Sire, Stephane; Bogdanov, Evgeny; Gillet, Denis; Wild, Fridolin (2010, to appear): Introducing qualitative dimensions to analyse the usefulness of Web 2.0 platforms as PLEs, In: International Journal of Technology-Enhanced Learning (IJTEL).

SF, SourceForge, https://ltfll.svn.sourceforge.net/svnroot/ltfll/Wp6/wp61/

Sporer, Thomas; Reinmann, Gabi; Vohle, F. (2007). Bologna und Web 2.0: Wie zusammenbringen, was nicht zusammenpasst? In: R. Keil, M. Kerres & R. Schulmeister (Hrsg.). eUniversity - Update Bologna. Education Quality Forum. Bd. 3. S. 263-278. Münster: Waxmann.

Wild, Fridolin; Sobernig, Stefan (2006): Interoperability Framework Draft for the Distributed Open Virtual Learning Environment, Deliver-able D3.1, iCamp Consortium.

Wild, Joanna; Wild, Fridolin; Kalz, Marco; Hofer, Margit; Specht, Marcus (2009): The MUPPLE competence continuum, In: Proceedings of the 2nd workshop on Mash-Up Personal Learning environments, EC-TEL 2009, Nice, France.

Wilson, Scott (2010): Wookie REST API. Available at: http://incubator.apache.org/Wookie/Wookie-rest-api.html, last access: 2010-11-25.

Wilson, Scott; Zuzak, Ivan; Hoisl, Bernhard et al. (2010): [#WOOKIE-133] Implement Inter-Widget Messaging. Available at: https://issues.apache.org/jira/browse/WOOKIE-133, last access: 2010-11-25.

Appendix A Widgets' IWC Descriptions

Below is a description of widgets created by all WPs along with their IWC data set. The GUID is used to uniquely identify a widget and locate it on the server (as the directory structure is based on the GUID). The IWC namespace the widget belongs to is stated as well as the format used to store data. This list should ease coordination of the long thread as everybody can see which data is set in a widget and which data a widget needs in order to do something.

WP4.1 LeaPos (Short Thread)

Widget name	Student Course Management
Description	Widget handles course, question, and answer management
GUID	http://www.l(tfll-project.org/widgets/WP4/student-course-mgmt
Namespace	WP4
Data format	JSON
Sets data	"method": method used for scoring algorithm (static) "content": student's answer to a question "question_id": integer specifying question id "log_id": integer specifying id for logging purposes
Listens to	

Widget name	Concepts Feedback
Description	Widget displays found, missing, and additional concepts according to a student's answer
GUID	http://www.ltfll-project.org/widgets/WP4/feedback-concepts
Namespace	WP4
Data format	JSON
Sets data	
Listens to	"question_id": displays a list of concepts (requires "content")

Widget name	Distinct Phrases Feedback
Description	Widget displays positive and missing distinct phrases according to a student's answer
GUID	http://www.ltfll-project.org/widgets/WP4/feedback-phrases
Namespace	WP4
Data format	JSON
Sets data	
Listens to	"question_id": displays a list of distinct phrases (requires "content")

Widget name	Scoring Feedback
Description	Widget displays a numerical score (percentage value) according to student's answer
GUID	http://www.ltfll-project.org/widgets/WP4/feedback-scoring
Namespace	WP4
Data format	JSON
Sets data	
Listens to	"question_id": displays answer score (requires "method", "content")

WP4.2 CONSPECT (Long Thread)

Widget name	CONSPECT
Description	All of the main functionality of CONSPECT takes place in this widget.
GUID	http://augur.w!u.ac.at/widgets/conspect
Namespace	WP4
Data format	JSON
Sets data	keywords
Listens to	feedurl

Widget name	CONSPECT feeds
Description	Takes the search term entered by the user and provide a list of feeds. When selected, pass the feed to the main CONSPECT widget.
GUID	http://augur.wu.ac.at/widgets/conspect
Namespace	WP4
Data format	JSON
Sets data	-
Listens to	feedurl

WP5.1 PolyCAFe (Long Thread)

Widget name	Conversation feedback
Description	Widget that displays textual feedback for the whole conversation.
GUID	http://www.ltfll-project.org/widgets/WP5/feedback
Namespace	WP5
Data format	JSON, XML
Sets data	“course_id”: the id of the current course “assignment_id”: the id of the current assignment “chat_file_id”: the id of the current conversation
Listens to	-

Widget name	Conversation visualization
Description	Widget that displays a graphical visualization for the whole conversation. Allows assessment of inter-animation and collaboration.
GUID	http://www.ltfll-project.org/widgets/WP5/ConversationVisualization
Namespace	WP5
Data format	JSON, XML
Sets data	“course_id”: the id of the current course “assignment_id”: the id of the current assignment “chat_file_id”: the id of the current conversation
Listens to	-

Widget name	Participant feedback
Description	Widget that displays textual feedback (indicators) for the each participant.
GUID	http://www.ltfll-project.org/widgets/WP5/ParticipantFeedback
Namespace	WP5
Data format	JSON, XML
Sets data	“course_id”: the id of the current course “assignment_id”: the id of the current assignment “chat_file_id”: the id of the current conversation
Listens to	-
Widget name	Utterance feedback
Description	Widget that displays textual feedback (indicators) and scores for the each utterance in the conversation.
GUID	http://www.ltfll-project.org/widgets/WP5/ UtteranceFeedback
Namespace	WP5
Data format	JSON, XML
Sets data	“course_id”: the id of the current course “assignment_id”: the id of the current assignment “chat_file_id”: the id of the current conversation
Listens to	“utterance_id”: the utterance that should be highlighted

Widget name	Enhanced search conversation
Description	Provides an enhanced search for chats and discussion threads, by ranking posts and participants according to the query.
GUID	http://www.ltfll-project.org/widgets/WP5/Search
Namespace	WP5
Data format	JSON, XML
Sets data	“chat_file_id”: the id of the current conversation
Listens to	“participant” (requires “score”): the score of each participant for the entered search query “utterance” (requires “score”) : the score of each utterance for the entered search query

Widget name	Assignment management
Description	CRUD management for assignments.
GUID	http://www.ltfll-project.org/widgets/WP5/AssignmentsManagement
Namespace	WP5
Data format	JSON
Sets data	“course_id”: the id of the current course
Listens to	-

Widget name	Conversation visualization
Description	CRUD management for conversations.
GUID	http://www.ltfll-project.org/widgets/WP5/ ConversationManagement
Namespace	WP5
Data format	JSON
Sets data	“course_id”: the id of the current course “assignment_id”: the id of the current assignment
Listens to	-

WP5.2 PenSum (Long Thread)

Widget name	PenSum
Description	All the student functionalities are concentrated in this widget
GUID	http://augur.wu.ac.at/widgets/pensum
Namespace	WP5
Data format	JSON
Sets data	rss feed url
Listens to	“doc_url”, “user_id”: adds (if possible) the doc pointed by doc_url to user_id’s profile so that he/she can synthesize it with PenSum’s help.

WP6.1 FLSS (Short Thread)

For the FLSS web services used in the short thread you may refer back to D4.3.1.

WP6.2a iFLSS (Long Thread)

Widget name	Search
Description	Widget can trigger a search for keywords
GUID	http://www.ltfll-project.org/widgets/WP6/search
Namespace	WP6
Data format	JSON
Sets data	“server”: URL to the service (static) “context”: ontology URI (static) “concept”: concept URI (static) “depth”: integer specifying depth of displayed graph (static) “lang”: two letter language code (static) “filterWeakPredicates”: Boolean - omits certain ontological relations when true (static) “keywords”: search term (set when user clicks on the search button)
Listens to	“keywords”: displays search term in input filed “node_name”: displays node name in input filed

Widget name	Definition
Description	Widget displays a definition (from Wikipedia) according to a keyword (search term or node name)
GUID	http://www.ltfll-project.org/widgets/WP6/definition
Namespace	WP6
Data format	JSON
Sets data	
Listens to	“keywords”: finds definition according to search term (requires “server”, “context”, “lang”) “node_id”: finds definition according to node id and node name (requires “server”, “context”, “lang”, “node_name”)

Widget name	Graph Visualization
Description	Widget displays a graph according to a keyword and can set a concept name when clicked on a node in the graph
GUID	http://www.ltfll-project.org/widgets/WP6/graph-viz
Namespace	WP6
Data format	JSON
Sets data	“node_name”: name of node in graph (set when clicking on a node) “node_id”: id of node in graph (set when clicking on a node)
Listens to	“keywords”: renders graph according to search term (requires “server”, “context”, “lang”)

Widget name	Scientific Papers
Description	Widget displays scientific papers (from Bibsonomy) according to a keyword (search term or node name)
GUID	http://www.ltfll-project.org/widgets/WP6/scientific-papers
Namespace	WP6
Data format	JSON
Sets data	
Listens to	“keywords”: finds scientific papers according to search term (requires “server”) “node_name”: finds scientific papers acc. to node name (requires “server”)

Widget name	Social Bookmarks
Description	Widget displays social bookmarks (from Delicious) according to a keyword (search term or node name)
GUID	http://www.ltfll-project.org/widgets/WP6/social-bookmarks
Namespace	WP6
Data format	JSON
Sets data	
Listens to	“keywords”: finds social bookmarks according to search term (requires “server”) “node_name”: finds social bookmarks according to node name (requires “server”, “concept”, “context”, “lang”)

Widget name	Social Slides
Description	Widget displays social slides (from SlideShare) according to a keyword (search term or node name)
GUID	http://www.ltfll-project.org/widgets/WP6/social-slides
Namespace	WP6
Data format	JSON
Sets data	
Listens to	“keywords”: finds social slides according to search term (requires “server”) “node_name”: finds social slides according to node name (requires “server”)

Widget name	Social Videos
Description	Widget displays social videos (from YouTube) according to a keyword (search term or node name)
GUID	http://www.ltfll-project.org/widgets/WP6/social-videos
Namespace	WP6
Data format	JSON
Sets data	
Listens to	“keywords”: finds social videos according to search term (requires “server”, “context”, “lang”) “node_name”: finds social videos according to node name (requires “server”, “context”, “lang”)

WP6.2b iFLSS (Long Thread)

Widget name	SNA Recommendation Widget
Description	Widget offers recommendations from your social network
GUID	http://www.ltfll-project.org/widgets/WP6/sna-recommendation
Namespace	WP6
Data format	JSON
Sets data	
Listens to	

Widget name	SNA Search Widget
Description	Widget displays resources in users social network files according to a keyword (search term or node name)
GUID	http://www.ltfll-project.org/widgets/WP6/sna-search
Namespace	WP6
Data format	JSON
Sets data	
Listens to	“keywords”: finds bookmarks (Delicious), videos (YouTube), presentations (SlideShare), and images (Flickr) according to search term “node_name”: finds bookmarks (Delicious), videos (YouTube), presentations (SlideShare), and images (Flickr) according to node name

Widget name	SNA User Search Widget
Description	Widget displays resources in users social network files according to a keyword (search term or node name)
GUID	http://www.ltfll-project.org/widgets/WP6/sna-usersearch
Namespace	WP6
Data format	JSON
Sets data	
Listens to	“keywords”: finds bookmarks (Delicious), videos (YouTube), presentations (SlideShare), images (Flickr) or users according to search term “node_name”: finds bookmarks (Delicious), videos (YouTube), presentations (SlideShare), images (Flickr) or users according to node name

Appendix B: Cross-Domain Interwidget Communication

This appendix presents the interwidget communication implementing a simple scenario: a widget sends to another widget a string of data and the second widget displays this data to the screen. This simple scenario utilizes Wookie-IWC fork server inter-communication capabilities and demonstrates a simple design pattern that can be followed during widget development

We create two separate widget folders with the same structure as the widget template. The first thing a developer needs to take care is to edit the config.xml file providing information on the widget (name, author, licence) and make sure that the polling function is active.

To configure that the two widgets are bound together, we edit the scripts/config.js file and provide a common namespace for both widgets:

```
var IWCnameSpace = "Demo";
```

On the first widget, we create a simple form with a send button.

```
<body onload="init()">
        <div id="ltfll_header"></div>
        <div id="ltfll_content">
            <form action="#" name="form" method="post">
                <input type="text" name="dataToSend" value="" />
                <input type="submit" value="Send"
onclick="sendData(this.form.dataToSend.value); return false;" />
            </form>
        </div>
        <div id="ltfll_footer"></div>
</body>
```

When *body* is loaded, the init() function (found in scripts/index.js file) is invoked. In it we set whether we want the LTfLL logo, footer or help to appear on this widget by calling the ltfll_design.Init() function with the corresponding parameters. This is done to provide a universal look and feel for all LTfLL widgets. Also we initialize the IWC communication by IWCinit() declaring whether we want the communication to take place using JSON(recommended) or non-JSON (simple key-value pairs) and if existing shared data should be initialized. In this case, we call IWCInit in IWCInit('JSON',true).

On the form created, we attach a click event handler to the send button that calls JavaScript function sendData with the value of the text box passed as a parameter. A good place to put this function would be index.js.

```
function sendData(data) {
      IWCsetVarJSON(IWCnameSpace, "DataVariable", data);
}
```

This function sets on the shared data a variable named DataVariable with the value the user input. In order to catch the data sent by the first widget, we add a call-back function (in scripts/iwc-callabacks.js) on the second widget that retrieves data and displays it on the second widget area. The function needs to be named IWC<keyword> where keyword is the IWC variable, in this case IWCDataVariable.

```
function IWCDataVariable(data) {
      document.getElementById("displayDiv").innerHtml = 'Widget 1
sent: ' + data;
}
```

The pattern explained above has been followed in a series of LTfLL' s widgets interactions.

1. "Search" widget communicates with "Definition" widget, which displays the definition of the keyword searched in search widget.

2."WP4 IWC Test 1" communicates with "WP4 IWC Test 2" exchanging messages (a trivial implementation of a chat service).

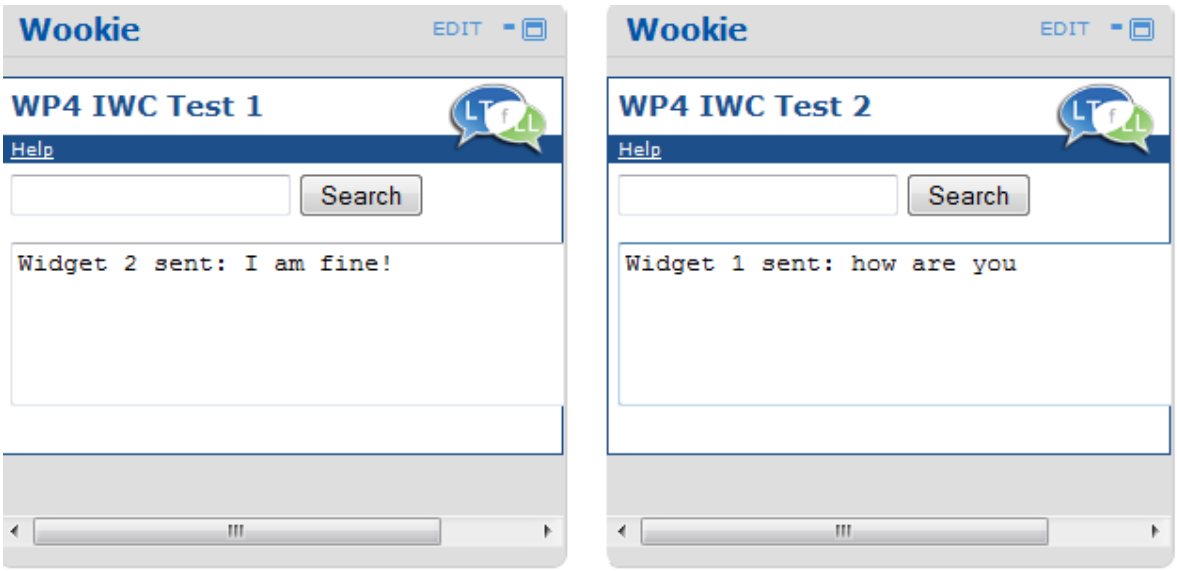


Figure 21. Example Cross-Domain IWC.

The CONSPECT widget follows a different format than the default LTfLL widget template. An iframe which consists of a regular widget served by the Wookie server (port 8888) is loaded by the widget container (in augur.wu.at test/development server the widget container is the Elgg platform plugin). In this iframe, an inner iframe is loaded which is served by Apache's default

port 80. This raises a technical difficulty to achieve interwidget communication because the client's browser considers the two iframes to belong in different domains. Thus, the same domain policy[1] restriction forbids requests back and forth.

To overcome this problem, we use a php script that acts as a proxy between the two domains. In this case when a widget wants to communicate with CONSPECT, the following sequence of events takes place:

1. Other widget wants to send something to Conspect. It invokes IWCsetVarJson(NS, key, val) and data is sent to the Conspect widget.

2. The Conspect widget triggers its call-back function and JavaScript function drop(data) is invoked. Drop executes a GET request (with data and openID of user as parameters) to proxify.php.

3. Proxify.php inserts a new entry to database table proxify

Field	id	openID	keyword	value	created
Type	int	varchar	varchar	varchar	timestamp
Description	auto-generated, used as a primary key	the openID identifier of the user performing the request	The variable name	The variable value	The time of the request

4. Iframe Conspect has a polling function (invoked every 1.5 sec) that performs GET requests to proxify.php and checks if there are new entries in DB

5. Proxify.php checks for new entries in DB (entries for the corresponding user made in the last minute). If yes, retrieves this entry and deletes it from DB

6. Proxify.php returns an xml document containing these.

7. Conspect gets the XML, manipulates it and updates its contents accordingly.

This method appends extra overhead to the communication (frequent queries to database, communication through the proxify script), however this was considered necessary because of the special nature of the CONSPECT widget.

[1] Same domain policy (or same origin policy) is a security feature implemented by all modern browsers that allows scripts to execute and access data originating only from the same domain. In this way malicious code is forbidden access to sensitive data (e.g. cookies of another domain) that could lead to session hijacking or identity theft.

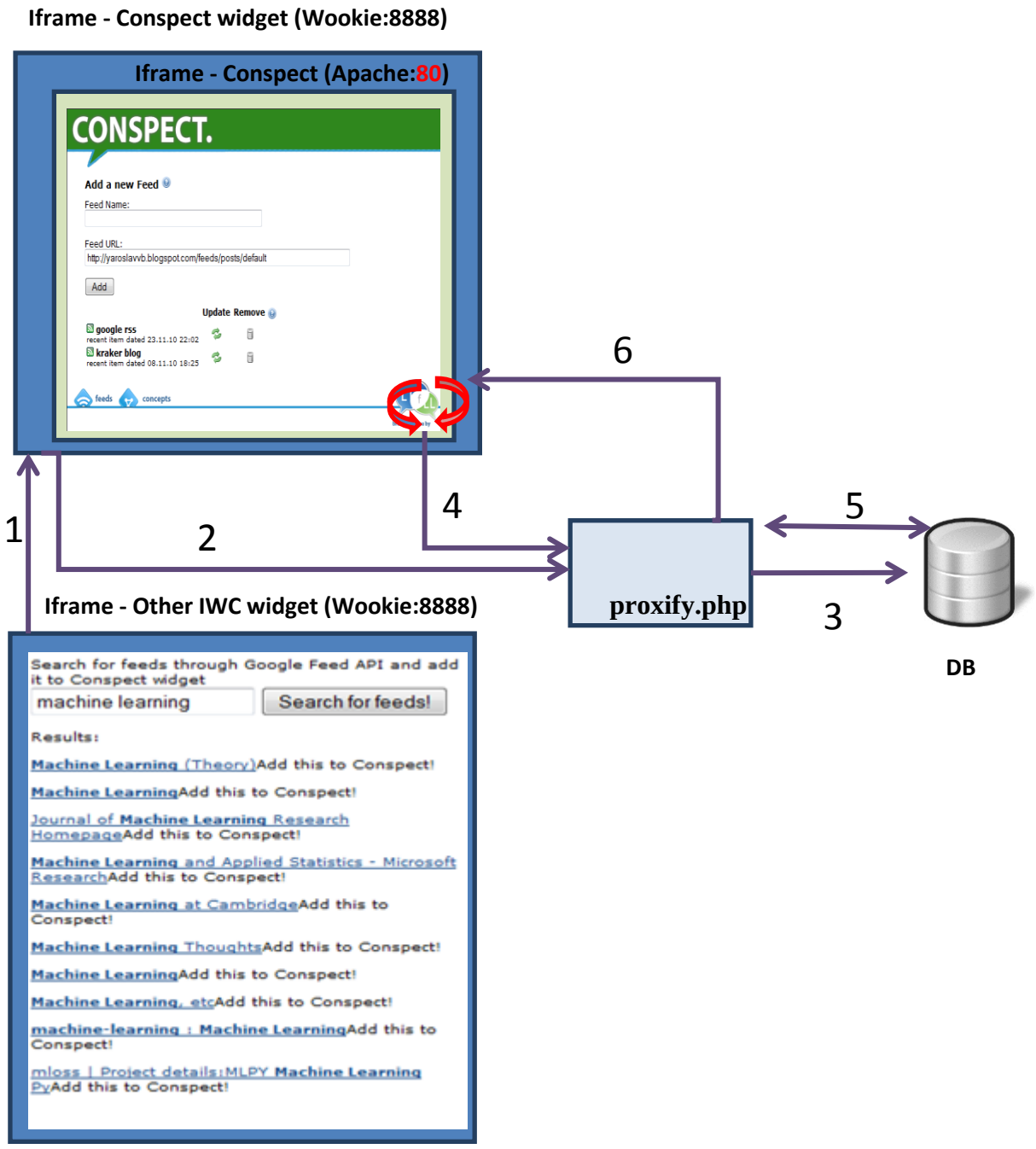


Figure 22. Communication in the cross-domain case.

Appendix C: Alternative arrangement of widgets

Personal learning environments (PLEs) are personal. So there are many ways to arrange the collection of widgets invented and developed in the LTfLL project, especially when combined with the steadily increasing number of use-case sized mini applications out there 'in the wild' (Google for example lists as of today over 140 thousand gadgets).

The applications developed within LTfLL, however, are aligned according to an underlying theoretical model in order to play well together. The selection depicted below provides an alternative to the long thread of useful arrangements of the LTfLL widgets, organised along activities performed in and in support of learning. It is an interim artefact that served to further discussion around the long thread.

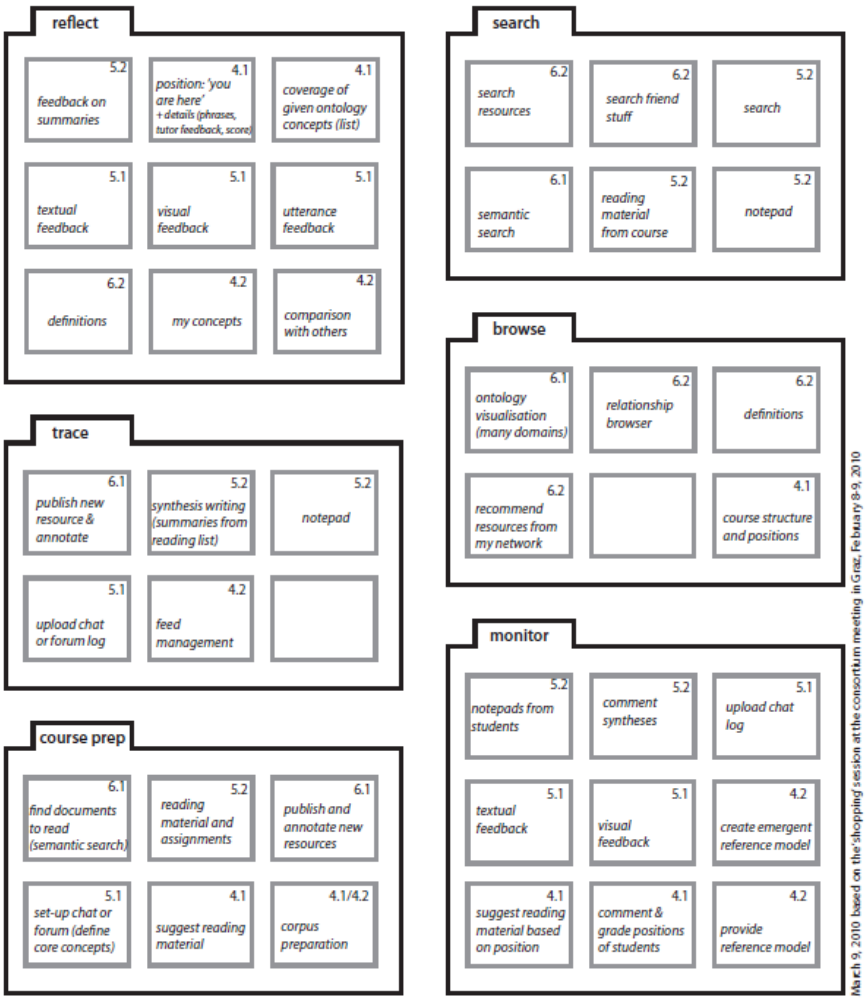


Figure 23. Activity-oriented 'tabbed' arrangement of widgets.