KP-LAB Knowledge Practices Laboratory – Specification of the shared space for knowledge practices software -release 1

Hannu Markkanen, Lili Aunimo, Merja Bauters, Vassiliy Tchoumatchenko, Ivan Furnadjiev, Tania Vasileva, A. M. Scapolla, Arianna Poggi, Jan Paralic, Frantisek Babic, et al.

27490

KP-LAB

Knowledge Practices Laboratory

Integrated Project

Information Society Technologies

D6.1. Specification of the shared space for knowledge practices software release 1

Due date of deliverable: 31/07/2006
Actual submission date: 10/09/2006

Start date of project: 1.2.2006 Duration: 60 Months

Organisation name of lead contractor for this deliverable: EVTEK

Revision [1.0]

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	PU
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Participants

Partner	Parther's short name	Participant	Email
	EVTEK	Hannu Markkanen Lili Aunimo Merja Bauters	hannu.markkanen@evtek.fi lili.aunimo@evtek.fi merja.bauters@evtek.fi
	TUS	Vassily Tchoumatchenko Tania Vasileva Ivan Furnadjiev	vasko@smi.tu-sofia.bg tkv@smi.tu-sofia.bg
	DIBE	Anna Marina Scapolla Arianna Poggi	scapolla@dibe.unige.it
	TUK	Jan Paralic Jozef Wagner Frantisek Babic	Jan.Paralic@tuke.sk jozef.wagner@gmail.com Frantisek.Babic@tuke.sk
	INPT	Pascal Dayre Hadj Batatia	pascal.dayre@enseeiht.fr Batatia@ipst.fr
	TESSERA	Thanasis Fotis	tfotis@tessera.gr

Version history

Version	Date	Author(s)	Description
0.0	29.7.2006	Lili Aunimo	First draft
0.1	31.7.2006	Jozef Wagner	Added Knowledge process tools
0.1.1	1.8.2006	Jan Paralic	Some text corrections/additions, Figure captions added
0.1.2	12.08.2006	Merja Bauters	Motivation Scenario
0.1.3.	16.8.2006	Hannu Markkanen	1. draft on requirements section.
0.1.4	21.8.2006	Hannu Markkanen	2. draft on requirements section.
0.1.5	21.8.2006	Lili Aunimo	draft on functionalities and architecture, especially portal and shared space management tools
0.1.6	29.8.1006	Hannu Markkanen	Added section 2.3.4 Linking. Elaborated requirements descriptions
0.1.7	29.8.2006	Lili Aunimo	Elaborated section 4 Architectural design, especially the parts Overall architecture and common services and Shared Space tool.. Added some comments and questions to KA and KP tools part in section 4.
0.2	30.8.2006	Jozef Wagner, Frantisek Babic	Extending KP tools sections. Added few comments.
0.2.1	31.8.2006	Jan Paralic	Corrections in TUK's input, commenting.
0.2.2	1.9.2006	Marina Scapolla, Arianna Poggi	Changes in 2.3.4 Annotations, 3.2 Knowledge Artefact Tool, 3.3 Annotation tool, 3.4 Other KA tools /editors, 4.2 Knowledge Artefact and Annotation Tools.
0.2.3	1.9.2006	Pascal Dayre, Hadj Batatia	Comments on integration with the domain model.
0.5	7.9.2006	Hannu Markkanen	Revised the section. Split the section "Annotating" to "Commenting" and "Adding semantic metadata". Amended section on non-functional requirements.
0.5.1	8.9.2006	Jan Paralic, Thanasis Fotis	Revision of sections on Knowledge processes.
0.5.2	8.9.2006	Marina Scapolla, Arianna Poggi	Revision of section on Knowledge artefact tool.
0.5.3.	8.9.2006	Hadj Batatia	Section Knowledge browser.
0.9	9.9.2006	Lili Aunimo	Final integration of partner contributions.
0.9.1	9.9.2006	Hannu Markkanen	Added Executive summary, Intoduction and Conclusions.
0.9.2	11.9.2006	Liisa Benmergui Patrick Ausderau	Proof-reading and layout

Table of Contents:

Executive Summary

This deliverable presents the high-level specification for the first release (M12) of the shared space for knowledge practices software, including the requirements, the functionality, as well as the service-oriented software architecture of the system. The requirements section describes the requirements process and the resulting high-level functional requirements. The functionality of the software is presented from the end-user perspective and divided into parts that form the major components of the architecture. The proposed architectural design introduces a number of components: the KP-Lab Portal and the portal level tools, the Knowledge artefact tools, the Knowledge process tools, the Shared space management tools, the core services and APIs, as well as the data layer.

1 Introduction

The objective of WP6, during the first twelve months of the project, is to design and implement the first release of the shared space for knowledge practices software, as well as and the common tools for KP-Lab knowledge practices. The specific research and development objectives have been elaborated based on the requirements that have evolved through WP6's participation in the design team work coordinated by WP2.

The functionality of the first software release is designed to support a limited set of educational scenarios that will experiment practices based on the trialogical learning approach. The trialogical learning happens within the frame of collaborative processes for advancing shared objects, which can be concrete or conceptual artefacts, or processes of advancing conceptual artefacts [Paavola et al 2004]. The learning process is characterised by the sustained work for developing shared knowledge artefacts across the learning process.

The WP6 software will provide the first KP-Lab experiments on trialogical learning in spring 2007 with a set of end-user applications that support the user activities of the scenarios in question. The software applications will allow participants to construct knowledge by modelling and visualizing actual objects and processes of work, as well as their relationships. The software is composed of a virtual collaboration space and a set of common tools that the user needs in carrying out the trialogical learning activities. This deliverable presents the high-level specification for this first software release (M12), including the requirements, the functionality, as well as the service-oriented software architecture of the system released in M12.

2 Requirements

The requirements process in the KP-Lab project has been organised according to the co-evolutionary design approach developed in WP2. In this section, an overview of the process is given, the detailed description of the co-evolution process can be found in [D2.1].

2.1 Overview of the requirements process and artefacts

At the core of KP-Lab's design approach is the idea of cyclic co-evolution of tools, practices and agents. Accordingly, the evolution of software tools will be carried out in several cycles providing progressively extended and refined functionality. Each cycle includes the elicitation and analysis of prevailing practices, the specification of pedagogical as well as technical requirements, the design and implementation of novel functionality, the testing and exploration of tools in practice, as well as the evaluation of the tools in the field trials. The document at hand covers the results of the first two steps of the first cycle of the co-evolution process.

In order to cope with the complexity posed by the diverse areas of application addressed in KP-Lab and the co-evolution of tools and practices, the project uses a flexible requirements engineering approach. Instead of striving for a complete a priori specification of the envisioned solution, the focus is on easy-to-create descriptions of the envisaged tools and practices that are continuously updated [D2.1].

The requirements process of the WP6 software release 1 began with the development of the educational and professional scenarios. This work was carried out within the Design Teams co-ordinated by WP2. In parallel, mock-ups were produced based on requirements elicited in the face-to-face and virtual workshops between some pedagogical and technical partners. User Stories based on the pedagogical scenarios and mock-ups were then written collaboratively by technical and pedagogical partners. Finally, Use Cases based on all of the above were produced by the technical partners.

The work has resulted in the artefacts listed below. These artefacts document the requirements and specify the functionality of the software system. The scope of the documents narrows towards the functionality of release 1 as the level of technical detail increases.

Design Principles

The idea of KP-Lab Shared Space is to support knowledge practices based on Trialogical Learning, which is being theoretically founded in the WP3. In the Kick-off meeting Kai Hakkarainen [Hakkarainen 2006a] presented twelve principles that have already been further elaborated in WP3. The design principles describe general characteristics and requirements of trialogical learning. They also aimed to give guidance for the technical development as general guidelines for design and criteria for evaluation (rather than as normative rules of design).
The design principles are the following [Hakkarainen 2006b]:

1. *Organise trialogical activity around shared objects*
2. *Interaction between personal and social levels of activity*
3. *Flexible tool mediation for trialogical activity*
4. *Fostering long-term processes of knowledge advancement*
5. *Development through transformation and reflection*
6. *Eliciting (individual and collective) agency*
7. *Cross fertilization of knowledge practices*

This list has also been used to support the pedagogical scenarios made for the courses that will be executed in the next academic year 2006-2007. Therefore, these principles have also indirectly affected the technical requirements and the process of writing User Stories and Use Cases. For more information on design principles, see [D3.1] and the description of design principles that is currently maintained on the KP-Lab project Wiki [WIKI-1].

Pedagogical and professional scenarios

Pedagogical and professional scenarios describe a concrete pedagogical or professional intervention and related activities in a particular context [D2.1]. While the KP-Lab design principles are grounded in theory, the pedagogical scenarios are closely bound to educational or professional practices and are situated in an organisational context. For the software development process, the scenarios provide the description of the activities that shall be supported by KP-Lab software.

In order to coordinate the work of the technical, pedagogical and theoretical work packages in KP-Lab, multi-professional design teams, co-ordinated by WP2, were established in the beginning of the project. The design teams were responsible for the development of the pedagogical and professional scenarios. For example, in the context of higher education context (WP8), 23 pedagogical scenarios have been developed in multiple domains, ranging from psychology, speech therapy, and medical education to education, as well as communication to media engineering, media technology and digital engineering. The pedagogical scenarios present a great diversity of learning activities and practices that the students, teachers, and experts are engaged in, and thus impose a tough challenge for the requirements analysis. In order to meet the tight schedule for producing the first WP6 software prototype, only a small subset of the scenarios were analysed for producing the current specification. The coverage will be extended for later releases of the software.

The current iteration of the scenarios explicates the first iteration of high-level technical requirements, both non-functional and functional, pedagogical requirements, and organizational requirements. The current set of pedagogical and professional scenarios is detailed in [DT-1].

User stories

User stories supplement the pedagogical and professional scenario by detailing the envisioned interaction of the user with the software system. The functionality required by the activities described in the pedagogical and professional scenarios is broken down into actions, each of which is covered by a separate story. User stories are kept short and simple in order to facilitate their easy maintenance and noticing any overlap between scenarios. They are written in plain English and are the product of the collaborative effort of the pedagogical and technical partners and provide a mediating artefact between them. See user stories for WP6 software release 1, currently maintained in a Wiki system [WIKI-2].

Mock-ups

A number of requirements mock-ups were produced to test design ideas and to elicit new requirements. The mock-ups were necessary to help in discussing the novel functionality in areas that the pedagogical partners found hard to envisage based on their prior experience. Mock-ups illustrate the screens and possibilities of user interaction and make the interactivity described by user stories more comprehensible. Similar to the user stories, mock-ups are used as mediating artefacts among technical and pedagogical partners.
Mock-ups are primarily produced by technical partners and reviewed and refined in close collaboration with pedagogical partners, mainly with WPs 3, 5, 6, 8, 9, as well as the Design Teams and Task Forces (for further information on the design process see [D2.1]). The process of developing a mock-up was iterative. Numerous half to one day workshops were organised during the first period of the project. Examples of mock-ups developed by EVTEK for the WP6 software release 1 can be found at [WP6-1]

Use cases

In contrast to user stories, use cases are meant to describe the envisioned interaction of the user with a specific tool in a more formal way than the user stories. Use cases are pertinent mostly to the technical partners and the main artefact to specify the functionality of the software and are therefore developed by the technical partners. Use cases for the release 1 of the KP-Lab shared space software are maintained in [WIKI-3].

Domain model

The shared domain model provides a preliminary design artefact, aimed to delimit domain related concepts and their relationships. It is directly linked to the technical development. An initial version of the shared domain model is available in [TechWP-1].

2.2 Motivating pedagogical scenario

This section provides a brief example of one pedagogical scenario and points out the trialogical aspects found in it in relation to the trialogical principles. The scenario presented here describes a course that will be executed in spring 2007. Most of the scenarios that will be executed during the next academic year (2006-2007) have trialogical features in them, but most of them do not have all the aforementioned features. Since the process is iterative, as was already stated, all the aspects/approaches/perspectives be they theoretical, pedagogical or technological, are evolving. Therefore, the prototype will also change, as will the requirements for the KP-Lab Shared Space during the project.

2.2.1Scenario description

This particular course was categorised as "Product-Oriented Learning". The full description of the scenario can be found in the Appendix II of [D8.1]. The course was chosen to be an example because it has worked as a mediating artefact between many partners in the development of the mock-up and prototype as also for the writing of the User Stories and Use Cases.

The goals of the course are to produce a multimedia product for a real customer, to learn how the project process that is related to a multimedia product's lifecycle is executed; and, to learn how to organise and manage the process in a team/group taking into account the customer's needs and the targeted users of the product. Work in the course is organized according to a project model, including team creation, planning, reporting, shared tasks and virtual collaboration, as face-to-face work. The deliverables the students are supposed to do are: the project documents, learning diary and the actual product that should be taken into use by the customer (company). All teams have a different project, including an outside customer. The following describes the main phases of the project process.

Phase 1: Initiation/"Organising the start"
The Initiation phase includes an introduction to project work and management, as well as an introduction to a "formal ontology of project work". The students are supposed to construct their own "ontology/semantic map" of the project work they do during the course. After each phase, the students compare their own "ontology/semantic map" to the "formal one" and discuss their findings.
The students form groups (done face-to-face in the lab). They try to find their customers independently. After the customer is found, a meeting with the customer is arranged to acquire information related to what the customer wants.
The students should decide the group roles they take responsibility for (e.g. manager, designer, coder, etc) and produce the synopsis of the product to be developed, including meeting memos of their meetings. Synopsis work includes producing a GANTT chart for the project, namely trying to envision what tasks are needed, how long would the tasks take, who is responsible for what, etc.

Phase 2: Planning
This phase includes: the "investigation" of what is already available and similar to what the customer wants (benchmarking); brainstorming of what the product is about; describing the

product and its needs; making scenarios; creating mock-ups including the graphs for the architecture; and, user-interface design iterations.

Phase 3: execution
The group must obtain the content from the customer and organise it. Often the group needs more intense contact with the customer during this phase. The group must implement parts of the content to the mock-up, ask for customer approval, test/evaluate within the team and with users. Finally, the group should report problem areas and possible manners to correct the mock-up or demos and continue the iterative cycles. In the last stage of the execuation phase, the group should implementing their code fully; i.e. adding actual functionality, or more refined and complex coding parts, polishing graphics.

Phase 4: Final evaluation/delivery
During this phase, the group must perform final tests with the product. The results are reported and the last corrections are made. The group must acquire the final approval from their customer and deliver the product to the customer. In conclusion, they must writing the final report of the project.

2.2.2Trialogical features of the pedagogical scenario

1. Organise trialogical activity around shared objects
A central idea of trialogical learning is that work and learning are organized around developing shared objects of activity. These shared objects can be conceptual artefacts, or collective activity systems or social practices, or products and product plans developed in companies [D 3.1].

The knowledge creation is intertwined with the product process. Collaborative knowledge creation (should) occurs when the team is working with the (shared) artefacts (i.e., mock-ups; test/evaluation plans, reports; deliverables, working on the product materials, (*products and product plans* [D 3.1]); and, continuously developing their "ontology/semantic map" (*conceptual artefacts* [D 3.1])).

Customer meeting memos are reused to organise the customer meetings and production of the meeting memos according to the team's way of acting (team's *collective activity system* [D 3.1]) and to develop and organise the activities between the team and the customer (*social practices* [D 3.1]). The social network can be developed further to meet the changing of the team's needs during the process (both product and knowledge process). For example, if the contact manner and intensity changes with the customer, it brings along different (new) *social practices.*
Parts of code are all the time reused across projects and developed further.

2. Interaction between personal and social levels of activity
The knowledge creation approach to learning is aimed at transcending the dichotomy between the acquisition approach on learning and the participation approach on learning. So trialogical approach concentrates on those processes where people are developing something new and combining individual initiatives and social processes for developing novel objects of activity [D 3.1].

For example, the mock-up, testing/evaluation and project process development are combining individual initiatives and the group's/team's, customer's initiatives, but also users' practices and needs have to be taken into account. The mock-up development starts in parallel design manner, namely every team member creates one mock-up of the product. These are discussed in the team and combined to a few (2-3) agreed mock-ups. These are shown to the customer for comments, which means another cycle of combining different perspectives. After this stage, the first tests and evaluations are done; thus, integrating the users' point of views and comments into the process. The project process development and execution includes: the practices that the customer has, the educational institution's practices, as well as the team's/group's practices. Obviously the former also affect the evolution of the GANTT as the mediating artefact of the project process.

3. Development through transformation and reflection

The emphasises is on developing through various forms of knowledge and between practices and conceptualizations, etc., meaning that transformations between tacit knowledge, knowledge practices, and conceptualizations are a driving force in processes of knowledge creation [D 3.1].

For example, the mock-up creation process is quite a lengthy one and definitely an iterative one. The team should be able to explain and reflect their choices in the mock-up design and the reasons for the changes they do. The team should share these explanations to enable collaborative understanding and knowledge on how the process of arriving to the "final graphical user-interface" has been accomplished. The management process needs explicit reflection. The team should reflect every now and then on how they work and how they have organised the management of the product process. For example, sticking to dead lines, changing them, creating or arranging the task, changing the roles of the team/group members etc. In addition, the ongoing construction of the team's/group's "ontology/semantic map" and the comparison to the "formal one" is a reflective long-lasting process.

4. Cross fertilization of knowledge practices

The KP-Lab system is meant for assisting people to solve complex, "authentic" problems and producing objects also for purposes outside educational institutions [D 3.1].

The search for a customer and contacting them, the design process with mock-ups, and team management issues are all "authentic" problems that involve outside educational institution contacts. Furthermore, the problems tend to be ill-defined and complex as the customers rarely do know what they want.

5. Flexible tool mediation for trialogical activity

The trialogical approach is based on the idea of mediation, that is, activities of human beings are mediated by tools, signs, artefacts and social practices, by which people can develop collaboratively and with cultural means. The trialogical approach has its basis on flexible tools that facilitate those aspects that are highlighted in other design principles, such as long-term, cross fertilized work around shared objects of activity which help an interaction between personal and social levels, and which help to make transformations between various forms of knowledge [D 3.1].

The mediating features of the artefacts (and signs) were already discussed above and the next sections describe the tools and functions derived from the requirements.

2.3 High-level functional requirements

This section summarises the high-level technical requirements elicited from the pedagogical scenarios for higher education [D8.1] developed within WP8 and from the requirements workshops organised between technical and pedagogical partners [WP6-2]. The high-level requirements are organised around the key concepts for the WP6 software. The key concepts were initially conceived during the proposal production time and have been elaborated during the requirements process.

It is important to keep in mind that the requirements have limitations as they were elicited in the context of knowledge creation practices in higher education. Other contexts such as teacher training or professional networks, also addressed in KP-Lab, might impose different requirements. The same limitation also holds for other settings in higher education. [D8.1].

2.3.1Shared Space

Shared space is a virtual collaboration space offering facilities for interacting with knowledge artefacts, knowledge process models, users and the shared space itself during a trialogical learning or working process. A shared space can either be personal space or a collective space. A collective space is created for the knowledge community involved in a trialogical process. The knowledge community can be formed around a group of people belonging to e.g. a project team, students attending a class, students of a university department, or any other type of collective.

A shared space provides the user with a configurable set of tools for

- working with the knowledge artefacts (e.g. creating, editing, storing, sharing, commenting, annotating semantically, disseminating, discussing)
- managing the knowledge processes (e.g. creating, changing and executing process models)
- managing the shared space itself (e.g. configuring the tools available)

Functional requirement	Description
Support both personal and collective spaces	In the personal space, the user who owns the space can work (e.g. create, modify, annotate, organize) with knowledge artefacts privately, and share them with their peers e.g. for feed-back, evaluation and discussion. The collective space contains knowledge artefacts that are subject to collaborative activities and that all space members have equal rights to. All group members can create or upload a new artefact into the space. The space can be configured to provide functionality to indicate the status of sharing (this functionality to be defined in later releases).
Moving artefacts between personal and collective space.	A knowledge artefact in a personal space can only be copied to a collective space, thus forcing it to be a shared artefact. On the other hand, in a personal space the owner of an artefact can create a link to a knowledge artefact in a collective space or copy it for his/her personal use.
Getting an overview of who is active at the moment in any shared space.	The system will provide an indication of what the current status of relevant users is (e.g. other members of a shared space). The status can be e.g. off-line, or connected and available for synchronous communication (with indication of the kind of connection), connected yet busy.
Exposing shared spaces as objects of collaborative activities	Shared space members should be able to work on a space as an entity, e.g. to annotate and discuss it. Shared spaces can also be associated with each other by using arbitrary types of relations.
Change notification service	A shared space provides an automatic, subscription-based notification service that informs users about changes (e.g. in knowledge artefacts) of the shared space.
Querying and searching the shared spaces.	Free text and semantic queries in the content of the shared spaces are supported.

2.3.2Views

User can browse and access the content of a shared space through *views*. A view is a graphical way of looking at the structure of information contained in a shared space, e.g. to visualize the relations of knowledge artefacts from different perspectives. A view portrays the knowledge artefacts and their associations in different arrangements, allowing people to view them in different manners. Views are basically directed and labelled graphs. The purpose of view is to aid the user to locate relevant knowledge within a context.

Functional requirement	Description
Visualizing the hyperlink structure in the form of a conceptual map.	The conceptual (map) view allows the user to browse across thematic or temporary relations between artefacts based on semantic metadata. The conceptual view is generated dynamically on the basis of metadata descriptions of the items within the view. Any item of the view can be selected as an anchor point to refocus the view. Hierarchical structures are possible.
Possibility to reorganize information in your own way providing various views to same knowledge.	User can create and save custom views, which are based on filtering mechanisms for choosing what is being shown. Artefacts can also be mapped on an image background as to provide visual support for indicating their content and relationships.
Displaying the selected metadata of artefacts.	Users can easily define what metadata is displayed in views.
Concurrent work by shared space members at different locations.	The system supports the temporal synchronisation of views open in different user agents. This means that changes in an artefact done by one user will be updated in other user agents' views that include the artefact in question.
Synchronised collaboration sessions within a group of users (e.g. members of a shared space).	The application and desktop sharing feature of the real-time communication platform implemented in WP4.

Note: Also other views will be built into the system. E.g. in T7.5 Knowledge practices analyzing tools will provide views that illustrate where and who has used which knowledge artefacts, which knowledge artefacts have been popular etc.

Figure 1. provides an example of an interactive mock-up created for visualizing the functionality of the shared space. The artefact view is in focus.

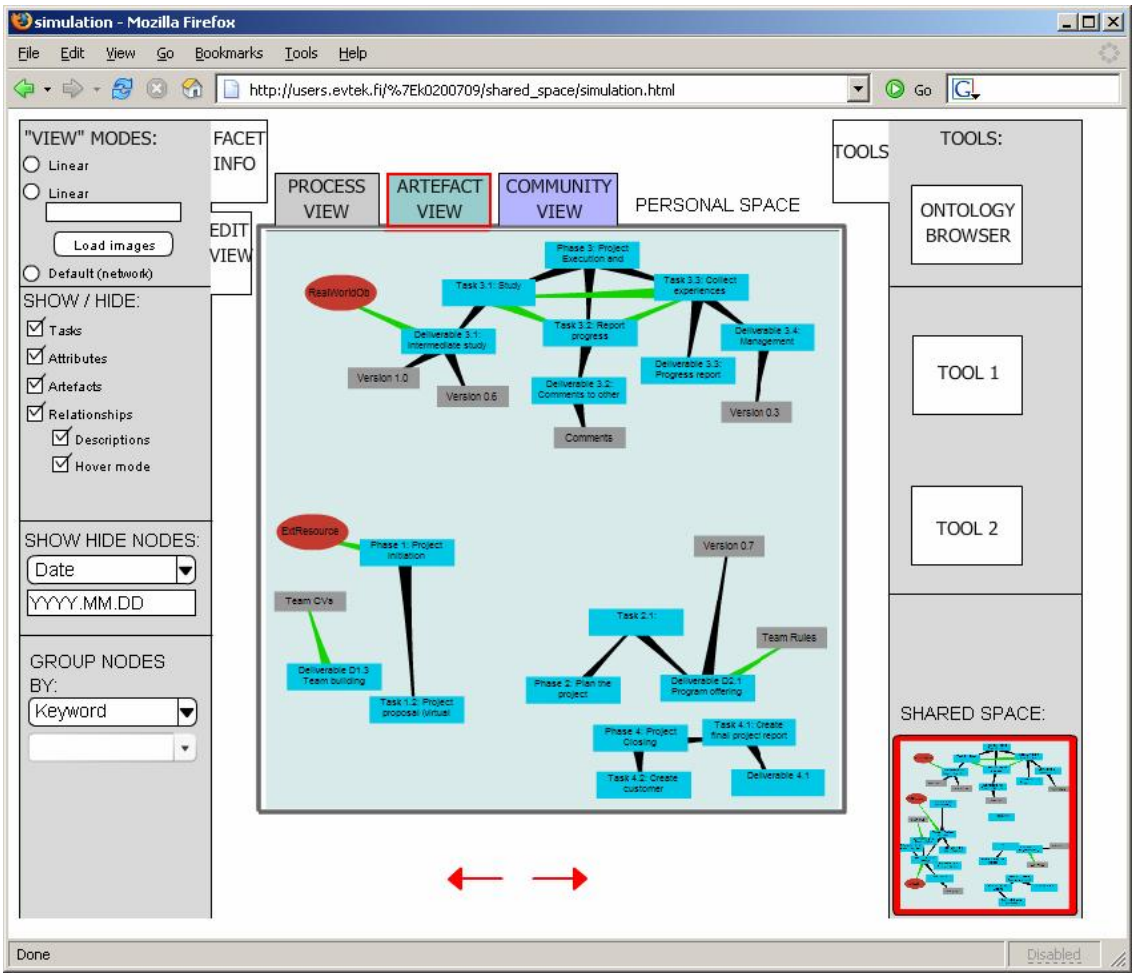

Figure 1: A mock-up for visualizing the shared space functionality.

2.3.3Knowledge artefact

A knowledge artefact (KA) is a semantically annotated information resource stored in a shared space. It is composed of the content element representing some explicit knowledge encoded and stored in certain media type, which is uniquely identified (e.g. by a URI), and of the semantic description (using a vocabulary defined in an ontology) of this content. A knowledge artefact is always located within one or more shared spaces.

Knowledge artefacts are produced and edited by the members of the shared space. For editing the content, see section 2.3.10 Content tools.

Functional requirement	Description
Collaborative production and editing of artefacts	In order to foster the collaboration on shared artefacts, the system shall support the creation and editing of artefacts such as: texts, presentations, visual images, tables, diagrams, and formulas. In case the system does not offer possibilities to work in some specific programs, it should then offer the facility to upload documents created in those programs.
Assigning artefacts to a group or individual.	In order to coordinate teamwork and to delegate rights, it is important to be able to assign artefacts both to groups and to individuals. This assignment is based on the user's membership in shared spaces. Assignment of artefacts to individuals or groups does not mean that the access should be restricted from others; assignment is for organizing the work in manageable views
Defining the access to and visibility of an artefact	There will be a possibility to restrict access to artefacts, but that should not be the first priority - trialogical processes are open by default
The status of all artefacts should be visible. (e.g. private, public, draft, waiting for review, reviewed, ready for publication)	The visibility of the status supports the coordination of activities. It is useful if users themselves can decide whether and when the produced artefacts is ready for publication; it gives a feeling of ownership. The vocabulary for status information can be user defined.
The development process of artefacts has to be comprehensible.	Creating artefacts is an iterative process and the necessity of building on previous versions is emergent. Also, the intermediate versions of the artefacts will be used as research data. The history of artefact evolution is visually represented in a way that allows the participants to navigate across the versions. A new version of a knowledge artefact is produced when the edited content is saved. Previous versions should be visible on demand.
Support of templates for creating documents.	Document-templates (e.g. for project reports) provide a means to pre-structure the document to be produced by the students and therefore provide a means for scaffolding.
Users shall be able to create, modify and reuse document templates. Templates should be supplemented with semantic information.	The collaborative creation and use of templates provides a means to foster the reflection on and generation of practices. Templates can be annotated (see annotation below).
Possibility to display several documents at a time.	The simultaneous display of documents eases the comparison of different documents.

Functional requirement	Description
Possibility to store/save synchronous communication sessions as artefacts (incl. deep links).	The recording of synchronous communication sessions helps participants trace back decisions and remedy misunderstandings.
Possibility to store/save asynchronous discussions as artefacts.	The recording of asynchronous discussions helps participants trace back decisions and to remedy misunderstandings.
Discussing knowledge artefacts or links between artefacts.	Discussion tool
The system shall allow up-load and storing external documents such as: PowerPoint slides, texts, data-records, videos and audio records.	In some cases, students might create artefacts outside the system; it should be possible to upload these artefacts. In order to foster sustained use of the materials produced, there should be a possibility for long term storage of documents.

2.3.4Knowledge process

Knowledge Process (KP) is defined as a set of activities conducted during trialogical learning or work, e.g. a set of activities conducted for a specific purpose, or a set of ordered steps across time intended to reach a goal or to produce a specific outcome.

Knowledge creation is one of the core aspects of trialogical learning and of the knowledge development in general (this includes also knowledge adoption, distribution, review and revision) within an organization [Bhatt 2000]. From the methodological point of view the knowledge creation processes have been studied in different contexts [Paavola 2006]. E.g. Carl Breiter's knowledge building approach has emerged from cognitive studies in the educational context. Yrjö Engeström's theory of expansive learning is based on Activity Theory, and Nonaka and Takeuchi's model of organizational knowledge creation originates from the analysis of work in Japanese companies.

The general challenge for describing knowledge processes are such that: all the participants should get an up-to-date understanding of the process; they should be able to make their individual and interconnected contributions to the process; and they should have the possibility to reflect on the course of process and practices of working together.

In the KP-lab software, process models represent knowledge processes. They are the basis for tools that provide support for joint reflection and development of work practices. These process models are represented as dynamic workflow (DW) models, which can be understood as a trialogical tool to visualize, negotiate on, construct knowledge and change the knowledge practices. DW emerged from traditional workflow systems.

Traditional workflow aims at automation of business processes, in which structure of the tasks and responsibilities for them are strictly predefined. The workflow system takes care of

execution and synchronization of tasks, and the information flows, to support individual tasks and how they are set-up. At present time, there exist some workflow systems standards, but many of them are still in development. One of main problems is complexity of workflow systems that causes the limited adoption of existing standards. No standards have been adopted on a broad scale.

Discussion that has been taking place in DT1 and later on in DT15 suggested that typical business workflows do not fully cover idea of trialogical learning. Several ideas about DW models were presented:

- Business workflow models (BWM) are not dynamic, they constrain activity and undermine creative intensions
- A contrasting opinion is that the time-scale is just another factor in the context
- We should keep process-structuring parts of BWM (i.e. showing time scale, interdependencies between deliverables etc., responsibilities, creation of documents, etc.).

DW should offer robustness to visualize the coordinated process and flexibly reframe the process. In such a way DW enables knowledge practices to be more flexible by combining the process, objects and people.

Based on the trialogical learning principles, six functionality criteria for DW models have been identified [Adams 2003]:

1. ***Flexibility and reuse*** means that at any point in time, there may be several possible sequences that can be followed, utilizing a subset of available actions to achieve the objective of the activity. Choices are dependent on the actual circumstances of the activity at that time (context). Thus, the model would need to manage a catalog of actions that, at runtime, could be chosen based on contextual information.
2. ***Adaptation via reflection*** means that a model should support evolutionary adaptation of processes based on the experience gained during each execution of the process. Plan adaptations for future instantiations can be achieved e.g. by recording the occurrence of deviations. Thus a plan is an artifact that contains history of its development.
3. ***Dynamic evolution of work practices*** requires, from the model, support of evolution of processes towards individual specializations without risking the loss of motivation for the overall activity.
4. ***Locality of change*** means that modifications should be able to be fully applied by changing a minimal number of components, and should impact minimally on associated components. One approach would be to support the definition of a workflow process as a set of sub-processes, each of which is a distinct entity representing a single action. Changes made within one sub-process (activity) will not impact the other sub-processes.
5. ***Comprehensibility of process models*** to all stakeholders, supporting representation at different levels of granularity. One possible approach is a hierarchical set of linked encapsulated sub-processes, i.e. each sub-process would be a (simpler) workflow model on its own.

6. ***The evaluation of exceptions to "first-class citizens"*** regards exceptions as events that provide an opportunity for a learning experience. Exceptions handling will be needed in both, design as well as execution phases. Selection of suitable actions from a repository of available actions could be made contextually dependent. If not precise action can be found, some approximate one can be selected and adopted into required context.

The "Multiprofessional teamwork and use of Dynamic workflow tool" pedagogical scenario [DT-2] identifies several requirements for DW tools:

1. The user should be able to define desired processes and how they should or could be performed. The description includes elements such as tasks (that can be decomposed to subtasks), schedule, milestones, resources used, allocation of work/responsibilities (division of labour).
2. The user should be able to trace and review what actually happens during a process, and view the status of the process. Possibility to create links between knowledge process model and the knowledge artefacts is very important.
3. The user should be able individually select the items he wants from Process, Artefact and People view into so called "Hybrid view". This offers flexibility to select, arrange and modify different elements of process. Hybrid view can be saved into user personal space.
4. New mixed artifact, for different users, can be created in the Hybrid view and made public. This will be an object of collaboration and links to resources that the artefact is based on are important.
5. The user should be able to join a discussion about each element in the process model.
6. History of all decisions or discussions about elements are saved, e.g. in Wiki pages.
7. User should be permanently informed about changes in the process model, e.g. internal messages or emails.
8. User should view his relevant tasks, personal list of tasks.
9. User should view relationships between particular elements of KP. The connections between KA and elements of KP are important.
10. The user should be able to change the model of the process online based on his custom rights.

Functional requirement	Description
Planning of the trialogical activities	User can define the process model in terms of work breakdown structure (tasks – subtasks), schedule, milestones, resources used, responsibilities. Labels should be user modifiable. It should be possible to provide structuring, but simultaneously facilitate emergency of novel lines of activity.
Reviewing activities	View the knowledge produced (different artefacts). Connection between KA and KP, see section 3.6. View the state of the process, e.g. what is the current working phase and indication of the phases that have to be done.

Functional requirement	Description
Modifying the planned activities on the basis of new demands and courses of action.	Users can make changes in the process model/ description on fly.
Definition of user rights	Different user rights may be defined on the level of the whole process as well as on particular tasks. In such a way the necessary flexibility in modification of processes on-the-fly may be achieved.
Assigning users to (sub)tasks	Each task can have several users assigned to it. This connection defines who is responsible for the task and should produce e.g. deliverables assigned to this task.
Linking of process model and knowledge artefacts.	User can create an arbitrary association (link) between a process model element and a knowledge artefact (e.g. meeting minutes). The linking is done in the similar way as linking of knowledge artefacts (see above). KA can be linked to specific task e.g. as a deliverable or just as a resource.
Commenting, annotating and discussing the process model.	User can attach comments (informal annotations) into any element of process model. Annotations (both formal as well as informal) of particular tasks or of the whole process are possible in the same way as by the KA annotation. User can start a discussion thread on any element of process model using e.g. Wiki or discussion tool.
Visualization of the process model.	Venn-diagram type of presentation should be used for presenting process model.
Visual modelling of progress (advancement)	System offers the possibility to check and track the history of a task.
Process (project) management views and tools.	Gantt-chart view. Visualization of progress linked to artefacts (e.g. “late of plan”, “need help”) The workflow system is in the background helping to manage time, coordinate efforts, and meet the milestones agreed.
A library of process model templates.	A process model can be created on the basis of a template describing the type of process to be implemented in a shared space. User should be able to store a template drawn from a process description created in a shared space.

2.3.5Linking

Linking is used for associating items of a shared based in some way. Linking structures helps participants organize and structure the knowledge in the way they want. The associations can be defined by using formal semantics defined e.g. by an ontology, or by informal text.

Functional requirement	Description
Interlinking of artefacts and discussions.	The direct interlinking of artefacts and discussions shall support to focus and ground collaboration. Thereby it is important that discussions can be bound to an artefact and that artefacts can be referenced in discussion
Organization of multiple artefacts in the form of a hyperlink structure (e.g. Wiki, concept map, argument map) or other general tools for working with the content of knowledge artefacts.	The creation of links between artefacts shall be as easy as possible. Hyperlink structure across various types of documents is supported.
It should be possible to classify nodes and arcs in the hyperlink structure based on a given or freely selectable ontology or metadata schema including a classification based visualization.	The classification of nodes and arcs based on an ontology allows to create scaffolds, but also to develop personal languages and grammars when needed.

2.3.6Commenting

A comment is some additional textual information associated with any (content) item of a shared space, such as knowledge artefact, a link (representing some kind of association) between items, and an element of a process description. Commenting a specific part of the content of an knowledge artefact should also be possible and will be a functionality of the content tool associated with the type of content in question.

Functional requirement	Description
Commenting items in the shared space.	Adding a comment as an informal annotation.
Commenting some part of the knowledge artefact's content.	The main content tool for the M12 release will be a Wiki that provides a commenting functionality.

2.3.7Adding semantic metadata

"Semantic metadata" is metadata that refers to a controlled vocabulary defined by a formal ontology. Adding semantic metadata will be referred to as (formal) semantic annotation later in describing the functionality of the annotation tool. Semantic metadata is important in creating a shared understanding regarding the meaning/purpose of certain knowledge artefacts and their relations and can be used to support many kinds of collaborative activities for

advancing knowledge; such as: giving feedback or discussing the contents of the knowledge artefact.

Functional requirement	Description
Adding semantic metadata to items in the shared space in a formal way.	The annotation tool allows users to select terms in a vocabulary defined by one of the available ontologies. For this purpose, services available in M12 SWKM prototype will be used, e.g. query and update services, or object exchange services (for details see [D5.1])

2.3.8Content tools for knowledge artefacts

A range of general tools for working with the content of knowledge artefacts need to be available in release 1 of WP6 software. The tools will be implemented mainly using existing open source software with required customization.

Functional requirement	Description
Creating and editing of documents that mainly contain text, drawings and images.	An existing Wiki with a good range of plug-in tools will be used.
Creating simple drawings.	Implemented using a plug-in of a Wiki system.
Discussion forum	Implemented using a plug-in of a Wiki system.
Blogging.	Implemented using a plug-in of a Wiki system.
Use of a concept-mapping tool for brainstorming, planning and knowledge modelling activities.	A loosely integrated, specialized concept mapping tool will be made available. If concept maps are required to be formalised as ontologies, the latter will be produced with an existing ontology editor and then exported manually to the semantic knowledge middleware (WP5).
Graphical visualization of concepts or argumentation.	A concept mapping tool will be used for visualizing the concepts and Map-It tool (WP7) is for argumentation.

For specific tools for working with multimedia, communication and e-meetings, see WP4 and WP7 specifications.

2.3.9Community

In trialogical terminology, community of practice is a group of persons with particular skills or expertise who interact formally within an organization, or informally in a network for shared pragmatic or knowledge-related goals.

For the KP-Lab software, a community can be defined as a group of people belonging to a particular shared space. The group of people can belong to e.g. a project team, students attending a class, students of a university department, or any other type of collective. The term “group” is often used as a synonym to community.

Social network refers to the social structure made of individuals or groups/communities in the shared spaces of the KP-Lab system.

User is an individual who uses the KP-Lab software applications.

Functional requirement	Description
Maintenance of users	Defining, modifying and deleting user information, roles, and access rights.
Defining groups (i.e. communities)	Done by defining who are the members of a shared space.
The system supports real time status of users.	The awareness of who is online helps to coordinate work and social navigation.
Visualization of the social network	Visualization composed of individuals and groups/communities and the relationships between them. Note: A rudimentary visualization is provided within WP6. Advanced network analysis tools are produced within T7.5.

2.3.10 Non-functional requirements

Non-functional requirements denote characteristics (properties and qualities) a technical system (tool) should have in order to be valuable and useful for those using it. Non-functional requirements include, but are not limited to, usability criteria or technical features such as interoperability, security requirements or other specific operating conditions.

This section summarises the high-level non-functional requirements elicited from the pedagogical scenarios for higher education, as presented in [D8.1], as well as the range of operational requirements to be considered. Most of the non-functional requirements are still on a generic level and will be elaborated to precise and measurable requirements as the user environments for the M12 release tests have been decided. The usability requirements will be based on the guidelines available from the Task Force Usability and the basic operation requirements will be gathered from the partner organizations that will participate in the field trials in spring 2007.

Requirement	Description
Integrativeness of tools	The different tools used within a given setting are integrated in the sense that they allow for smooth transitions between activities performed inside different tools. Data can be copied and moved easily between tools.
Easy to use tools	New users do not need to learn complex manoeuvres to be able to navigate, open, edit or manipulate objects. Tools have to be able to be use in extremely simple and obvious ways.
Deep customization	The user can decide which tools to use within a certain context. In some cases, deep customization might be restricted to the teacher or organizer of the course. It has to be possible to integrate discipline-specific tools if needed.

Requirement	Description
Secure Access	Access to the system and data transfer shall be secure. Security issues shall be transparent to the user.
Interoperability	The system must be runable by different browsers and operating systems. The system shall also support different input and output devices, such as mobile phone or PDAs.
User support	What kind of documentation, training, online help, and help-desk services are required?
Performance and quality of services	Specificy the speed at which part of a system performs under a particular workload. What are the maximum response times and capacity requirements. Requirements for system availability, such as what is the maximum acceptable time for restarting the system after a failure, what is the acceptable downtime per day?
System maintenance	How often will the system be backed up? Who will be responsible for the back up and for restoring the system after a system crash? Who is responsible for system installation? Who will be responsible for system maintenance?

3 Functionality

This section describes the functionality of the KP-Lab Portal and of the three sets of tools that it contains from the end-user perspective. The first section describes the functionalities available at the portal level, and the subsequent sections describe the functionalities of the three sets of tools, namely the Knowledge Artefact (KA) tools, Knowledge Process (KP) tools and the Shared Space (SS) management tools. The KP-Lab Portal prototype that will be implemented by M12 will be evaluated in field trials. These field trials are described in WP8 and WP10, among others.

3.1 The KP-Lab Portal

The shared space for knowledge practices that provides access to the three groups of tools and to the portal level tools is called the KP-Lab portal. Here the word portal is used as a general term, and it does not refer to a portal server such as Jetspeed. An example of the set of functionalities that the KP-Lab portal could provide is shown in Figure 1. The functionalities (or tools) that the portal provides to the user are:

a. Login/logout. The login tool is used to perform user authentication. The user credentials are checked against the user database or the directory service at the user's home organization once and after that the user can access all tools without having to provide his user credentials again. The Single SignOn (SSO), attribute exchange across organizational boundaries, and the management of identity and access permissions will be done using the Shibboleth software [AHM06, SHI06] or the Liberty Alliance software. If a user belongs to several organizations, he has to choose one from the list provided in conjunction with

the login tool. He may naturally log into several separate instances of the KP-Lab Portal simultaneously if he wishes to use all of his usernames at the same time. The use case for the login tool can be found at [WIKI-4].

b. Management of user preferences. User preferences contain information such as the visual settings of the portal and portlets, user nickname, default tools available as portlets and possibly a default shared space to be opened when the user logs in. The user naturally has to be logged in in order to be able to modify his preferences.

c. Shared space browser. This might mean a list-based and/or graph-based browser, see the boxes pointed to by arrows in Figure 2. Figure 3 illustrates an example of a graph-based shared space browser. Figure 4 illustrates an example of a shared space browser that can be used to browse also KAs and KPs related to a shared space. The browser also allows the user to specify the radius of the shared space network to be viewed.

d. User statistics viewer. When the user is not logged in, the user status tool could show for example only how many users are online in the portal and when the user is logged in, it could show more specific information.

e. There might also be other functionalities (or tools or portlets) such as a notifications tool for portal wide news and other issues, a global chat tool, a user specific history viewer that shows information such as the most recently visited Shared Spaces of the user, etc. The user preferences management portlet could be used to specify which of these tools are shown to the user.

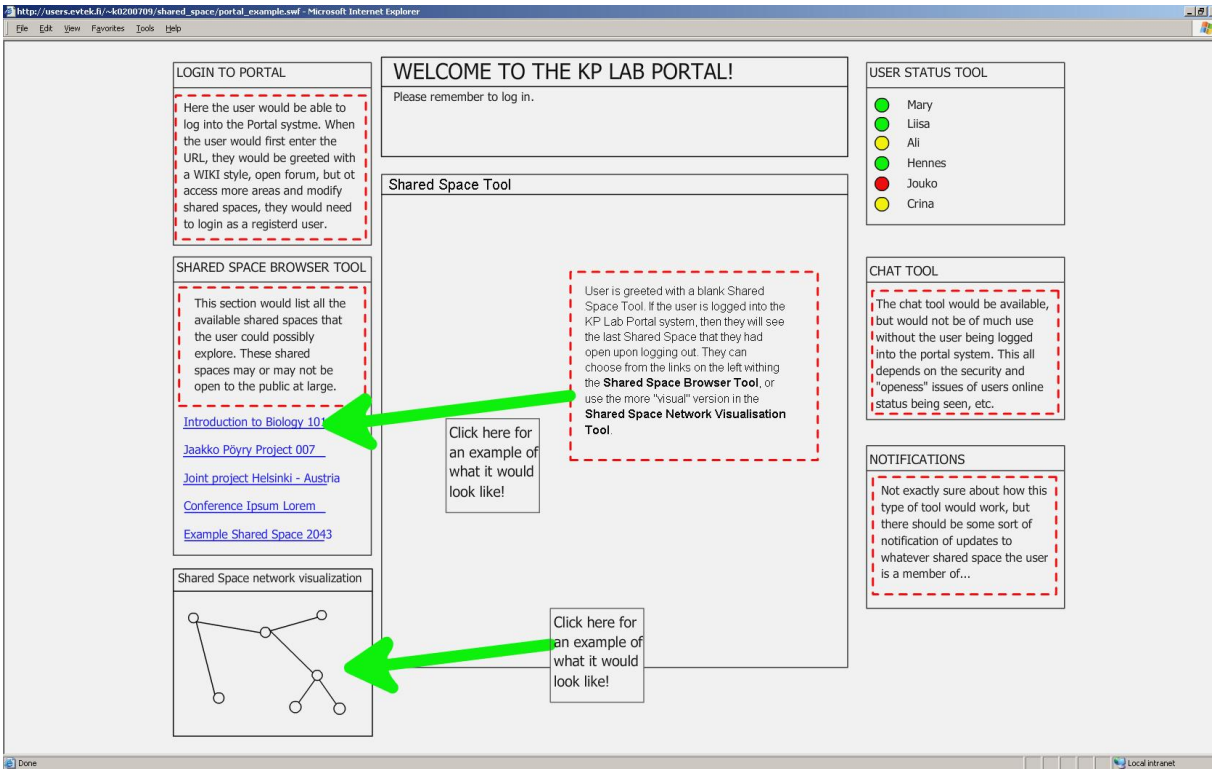

Figure 2: An example of the functionalities available in the KP-Lab Portal.

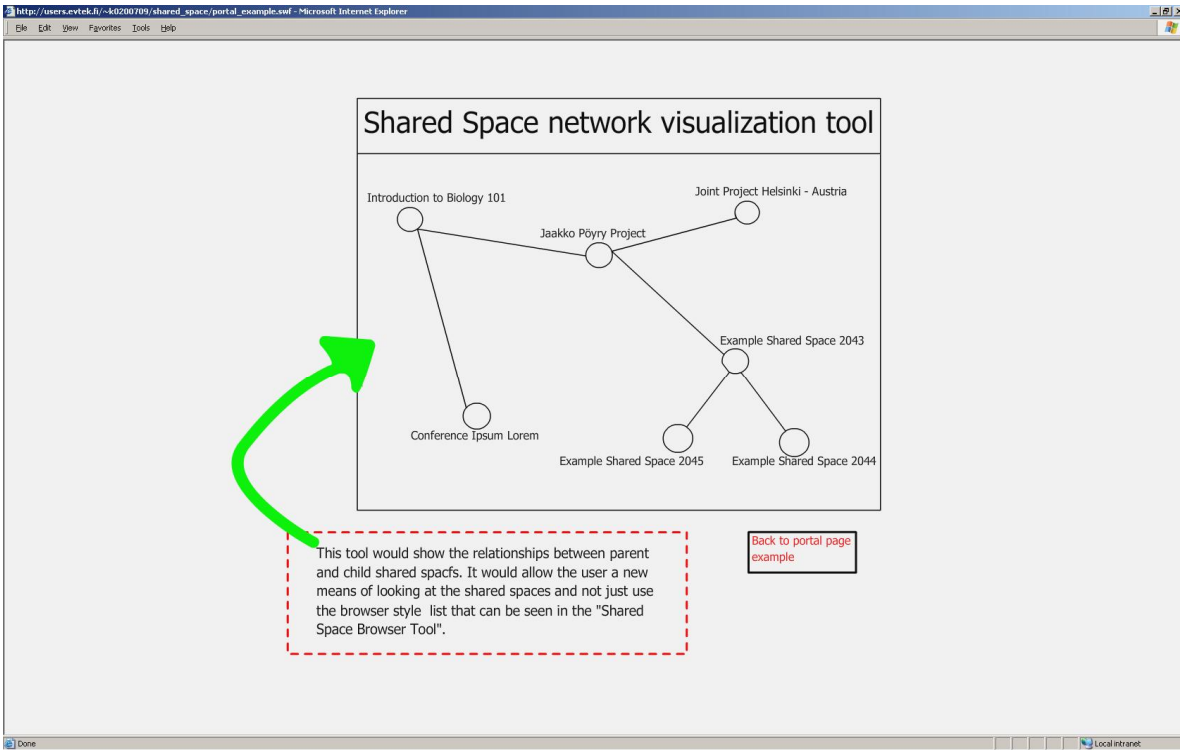

Figure 3: An example of the graph-based shared space browser.

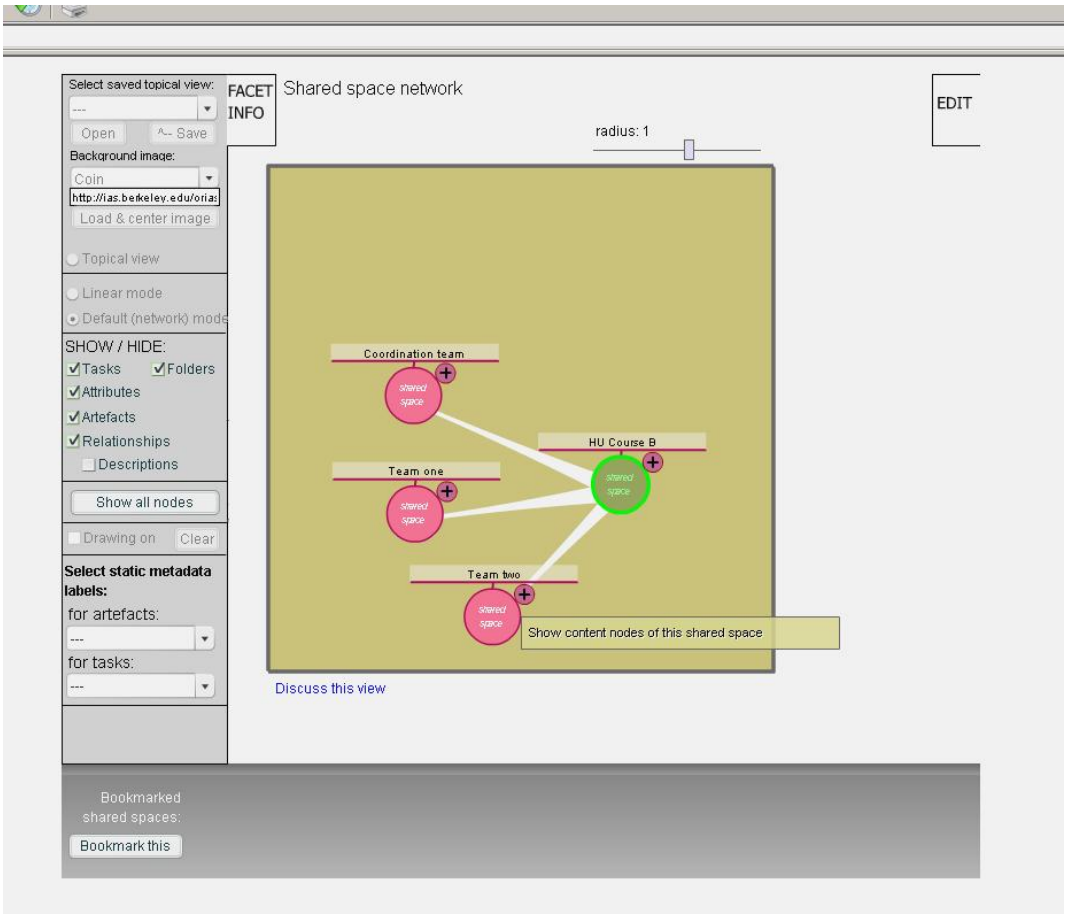

Figure 4: An example of the shared space browser with support for showing also KAs and KPs and with support for defining the radius of the shared space graph.

3.2 The Knowledge Artefact Tool

The principal role of the Knowledge Artefact tool (KA Manager) is to create, modify, save and delete knowledge artefacts. To accomplish its tasks the KA Manager collaborates with:

- the shared space management tools/services
- the content repository for storing, retrieving, deleting the KA content;
- the knowledge repository for storing, retrieving, deleting the KA metadata/annotations.
- the knowledge repository browser for retrieving KAs.

The KA Manager should perform the following functions:

- Create new KA in the shared space.
- Retrieve and update existing artefacts
- Delete KA
- Query the knowledge repository

- Retrieve KA from the persistent storage and transfer them to the shared space (making use of services available in M12 SWKM prototype, e.g. query or object exchange services, see [D5.1])
- Save the KA into the persistent storage
- Invoke the content editor specific to the artefafact media type that must be created or visualized.
- Notify changes to the synchronization service.

Other functions (not to be developed for the M12 prototype)

- Versioning support for the content repository
- Collaboration between the logging and profiling tools

3.3 Annotation Tool

We define a very simple annotation taxonomy to determine what kind of annotations will be handled by the annotation tool and where these annotations will be stored.

Formal vs. informal annotations

Formal annotations are annotations where subject and predicate are Uniform Resource Locators (URIs), and the object is an URI or a formal literal. Formal annotations are referred to as **semantic** if based on ontological terms and definitions. They are stored in the knowledge repository.
Simple comments on any item of the shared space can be considered formal annotations on condition that the subject of the comment can be defined by an URI. (The subject is the item URI, the predicate is "dc:comment", the object is a formal literal).
The annotation of links, which have been established by the knowledge process tool, would from the technical point of view mean annotating Resource Description Framework (RDF) statements. We would use reification to produce an URI for the statement. If we annotated links i.e. RDF value names, then the annotation would refer to that particular RDF value name in all statements that use it.

Informal annotations are related to items inside the artefact (e.g. particular paragraph in text document, or particular image area) and are stored inside the digital object (content item) in the application-specific format. An example of such a case is the notes inserted in the documents by the word processors like OpenOffice.org Write or MS Word.

From the end user point of view, informal annotations are available only opening the content item in the proper editor, while formal annotations require different user interfaces. For example, the "Add comment" menu choice, which is present in the mock-up of the shared space for item annotation, can be followed by typing a comment in the string format. If the annotation requires to upload an ontology, then the process becomes more complex and so far, the user interfaces are not available.

Summarizing the Annotation tool should perform the following functions:

- Visualize annotations
- Create, update, delete annotations

To accomplish its tasks the Annotation tool collaborates with:

- the knowledge repository browser for retrieving KA annotations
- .the knowledge repository for storing, retrieving, deleting the KA annotations.

3.4 Other KA tools / editors

Specific tools for creating manipulating and editing content with different file formats and MIME types, wikis, text documents, chats, forums, etc

The M12 prototype will implement a wiki-based editor for content editing

KA mapper - mapping artefacts to background images. Approaches for storing the mapping:

- the background image URI and the artefact coordinates (#x,y,z,t) are represented as properties in the content repository.
- the background image and artefact(s) coordinates are part of the shared space profile of the team

3.5 Knowledge Process Tools

Knowledge Process (KP) tools will provide a set of functions and interfaces necessary for creation, management, and annotation of knowledge processes composed from elements.
KP tools functions:

- Create, view, update and reuse a task: User can use existing templates or can create a new model of a task. This task can be used more than once and in different processes.
- Set-up description of the task, see Figure 5. Characteristics of the process (subtasks, division of labor, resources etc.) are defined by this description and can be visualized using the Gantt chart.
- Set-up relationships between tasks: Relationships between tasks are defined in Gantt chart by vertical lines and by slots called prerequisites in task description. “Prerequisites” refers to tasks that need to be completed before the actual task can begin.
- Execution of a process: User can follow current state of a process by timeline, making use of attributes of particular tasks (these attributes are set either manually by responsible users of particular tasks, or automatically via some actions).
- Change task settings on the fly: User can flexibly make changes in description of the relevant task based on their user rights. E.g. author of the full process (e.g. teacher) can make interventions that change the overall process, its duration or goals. Other users can make changes on the local level, e.g. within the task which they are responsible for.

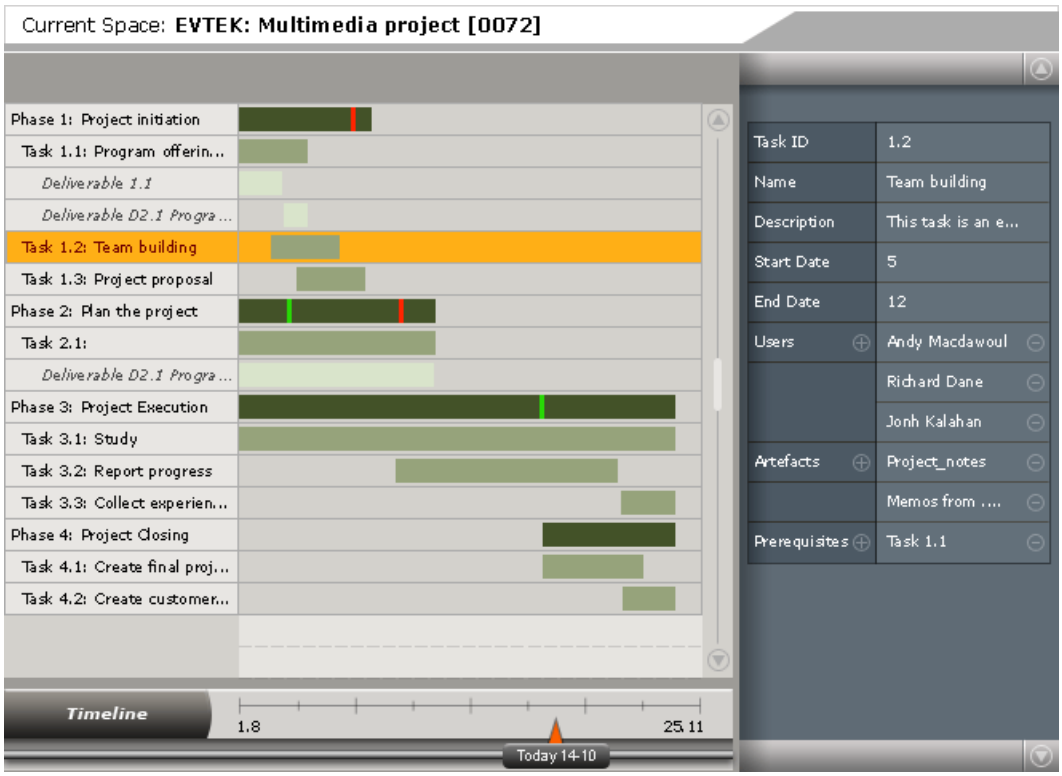

Figure 5: Example of process view

KP tools functionality will be accessible to the users via different portlets and or via API, see architecture of the Knowledge Process Tools in Figure 13.

1. Knowledge process view portlet will display a knowledge process and its content and allow users to browse within the given knowledge process. This portlet will provide the following functionalities: view and update the knowledge process; create structure of the knowledge process.
2. Knowledge process description/annotation portlet will allow user to describe/annotate a knowledge process in a way similar to describing/annotating a knowledge artefact in KA tools.

Knowledge Process Tools will provide an API for working with knowledge processes. This API can be used by other parts of shared space or by other KP-Lab tools. E.g. for Share space tools that will provide deleting or browsing capabilities of existing knowledge processes can use the KP tools.

Some tasks will have several knowledge artefacts connected to it (e.g. as deliverables or learning materials). This connection is displayed in process view. On the other side, a Knowledge Artefact can be associated with several tasks. This is for example shown in artefact view of the shared space. If this bi-directional connection is represented on both sides, i.e. with the knowledge artefact, as well as with the task, care should be taken to ensure integrity of these connections. There is also a possibility to store this connection only in the process description and query process tools for tasks related to given artefact.

3.6 Shared Space Management Tools

The Shared Space Management Tools offer to the user the following functionalities:

- creation and deletion of a Shared Space,
- annotating a Shared Space or a link,
- managing the information related to a Shared Space,
- managing the users, their roles and their access rights in the context of a Shared Space,
- managing the tools available in a shared space.

These functionalities will be described in more detail in the following subsections.

3.6.1 The Shared Space Creator

The Shared space creator checks if the specific user has sufficient rights to create a shared space and if he has provided all the mandatory information for the new space. If these prerequisites are fulfilled, it creates a new shared space and stores it into the Knowledge Repository. The required fields for a new shared space are the following:

- shared space name and
- the manager member(s). The default value is the creator of the space.

The shared space creator is a very simple tool, and the idea is that the shared space manager will be used to further configure the shared space. The shared space creator is context sensitive in the sense that if it is used while the user is in a specific shared space, the new shared space will be a child space of that space. This means that it will inherit all the information concerning that space. On the other hand, if the shared space is created while the user is not in any specific shared space, the shared space will have no parent shared space from the point of view of the user. The user may afterwards use the shared space manager to add parent(s) to the shared space. From the technical point of view, all the parentless shared spaces are allocated a common parent node. It may be called, for example, the root node. This root node is required in order to form one single graph from the shared spaces.

The use case diagram for the Shared space creator tool is available at [WIKI-3]

3.6.2 The Shared Space Annotator

The Shared Space Annotator is used to annotate a given Shared Space with KAs, KPs, other Shared Spaces, arbitrary literals and terms listed in a predetermined vocabulary. Also annotations themselves may also be annotated.

The annotation is performed using an arbitrary literal if the annotation is small, i.e. if the annotation is a couple of words only and it does not belong to any predefined vocabulary. If the annotation is large, it is performed creating a new KA. The small annotations consisting of arbitrary literals are entirely stored in the Knowledge Repository and the large annotations are stored both in the Content Repository and in the Knowledge Repository. The actual content, i.e. an Open Office document, is stored in the Content Repository and the URI that points to the document is stored in the Knowledge Repository. The user may see the action of creating

small and large annotations as commenting. A Shared Space may also be annotated using terms listed on a predefined vocabulary. This vocabulary is stored in Resource Description Framework Schema (RDF-S) format in the Knowledge Repository. Finally, a Shared Space may as well be annotated with an existing KA, KP or another Shared Space or with a copy of an existing KA, KP or Shared Space. An annotation is always a directed edge between two nodes. The edge begins from the Shared Space node and points to the arbitrary literal, term listed in a vocabulary, KA, KP or other Shared Space.

From the point of view of the user, annotating shared space with a Knowledge Artefact or a Knowledge Process may mean bringing the Knowledge Artefact or Knowledge Process into the Shared Space or linking the Shared Space with the KA or KP. This may mean first creating a totally new KA or KP from scratch or creating a new KA or KP by making a copy of an existing KA or KP. The tasks of creating new KAs and KPs are handled by the KA Tools and KP Tools, respectively. Annotating a KA or a KP to a Shared Space may also mean sharing a KA or a KP with one or several Shared Spaces. In this case, no creation of new KAs and KPs takes place.

Annotating a Shared Space with another Shared Space may or may not imply modifications in the child shared space. If it does not imply changes in the child Shared Space, the semantics of the annotation is that the user wishes to make a loose relation between two shared spaces. The relation itself could be annotated with phrases such as “see also” or “the same project manager participates here also”. If annotating a Shared Space does imply changes in the child Shared Space, the annotation means that the target (i.e. the child Shared Space) of the edge inherits the parent Shared Space. More concretely this could mean that the child Shared Space inherits by default everything from the parent Shared Space and that the user could then specify items to be excluded from the inherited ones. As the Shared Spaces form a directed graph, a child space may have arbitrarily many parent nodes and they may contain conflicting information. For example, a username may have a different permission profile in different parent spaces. Which permission profile should the child space inherit? One solution would be to write rules to resolve the conflicts. In this case an example rule could be to give the username in the child shared space the minimum of the conflicting permission profiles and notify the end user about this. Another solution would be to just detect the conflict, leave the conflicting information blank and inform the end user about this. Figure 6 illustrates an example of the Shared Space Annotator Tool.

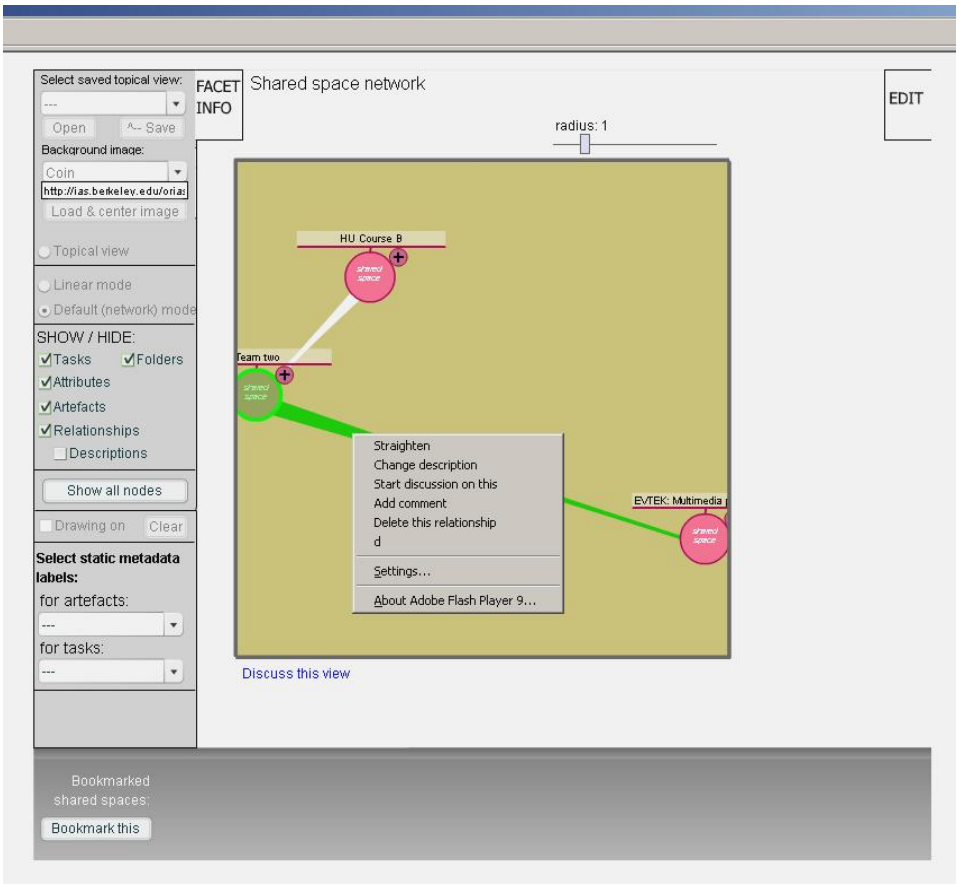

Figure 6: An example of the Shared Space Annotator functionalities. Using the Annotator, the user has the possibility to annotate a directed link with a literal or a KA, i.e. comment it.

The Shared Space Annotator may be used to annotate either Shared Space or a link with a literal or a Knowledge Artefact. A literal is used if the annotation is small, i.e. a couple of words, and a KA is used if the annotation is large, for example a whole word document. The small annotations are entirely stored in the Knowledge Repository and the large annotations are stored both in the Content Repository and in the Knowledge Repository. The actual content, i.e. the Word document, is stored in the knowledge repository and the URI that points to the Word document is stored in the Knowledge repository.

3.6.3 Information Manager

The *Information manager* is used to create, change and delete any information related to a specific shared space.
This information consists of the obligatory information that was listed above in conjunction with the description of the shared space creator (the shared space name and the manager member(s)). In addition, at least the following information may be created, deleted and modified:

- Free text description of the space
- The tools available in the shared space.

- Level of the shared space. Some typical examples include: individual, group, team, class, course, project, department, institute, school, university, company, and community.
- Time to live e.g. the start and/or end date of the space.
- Default access rights for different user groups or user organizations and/or for different roles. For example, only users who belong to the organization Evtek, who alos belong to group Media Technology Project and who have the role student may link and annotate KAs, KPs and SSs within that shared space.

3.6.4 User Manager

The *User Manager* is used to manage user information that is specific to the shared space in question. The user specific information is thus different than the one maintained at the KP-Lab Portal level. For example, a user may have the role student at the KP-Lab Portal level, but the role assistant in a specific shared space. The information manager is used to set general default access rights for a space, and it may, for example, be done by a secretary. The user manager tool is typically used by the teacher or project manager as the information managed is very specific.
The user manger is used to:

- Add or delete usernames of the KP-Lab Portal into a space based on their organization (e.g. INPT, Evtek), group (e.g. course on media technology) or role (e.g. teacher).
- Change username information in this space, i.e. change the role, group, or organization of a username and tailor access rights.

3.6.5 Tool Manager

The *Tool manager* is used either to enable, disable, add or delete tools in a shared space. The only tools that are by default available in every shared space are the KA tools and the shared space management tools. Other tools of the KP-Lab Portal may be disabled or enabled. Examples of such tools are: the process tools, video conference tools that enable sharing of screens and C-map. Some tools, such as the process tools, are tightly integrated into the system. Some tools might be available as services and they could be displayed in a portlet. The Sakai plug-in architecture illustrates this approach. Some tools are not integrated at all – they have only been installed on a server. C-map is an example of this kind of tool. A C-map server will be installed at Evtek and it will be offered as a separate tool to the end users. Adding and deleting tools from the shared space means managing arbitrary tools that are not part of the KP-Lab Portal. An example of such functionality is the possibility to add a link to any tool that each user has to download and install himself.

The use case diagrams for the Shared Space Management Tools are available at [WIKI-3] and [WIKI-5].

4 Architectural design

This section describes the architectural design of the KP-Lab Portal. The first subsection presents the global view and shortly introduces the components that will be described in more detail in the subsequent sections. First to be described are those components that are by default available in the KP-Lab Portal (Section 4.2, The Portal Level Tools), that constitute the Core Services and APIs (Section 4.3) as well as the database management systems (Section 4.4, The Data Layer). After that, the different sets of tools are described. These sets of tools consist of the KA tools (Section 4.5), the KP tools (Section 4.6) and the SS management tools (Section 4.7).

4.1 Overall architecture

The shared space for knowledge practices that provides access to the KA tools, the KP tools, the SS administration tools and the portal level tools is called the KP-Lab Portal. Each of the tools included in the KP-Lab Portal is a stand-alone tool meaning that it does not require any other tool in order to function. However, there may still exist interaction between the tools that are in the portal. For example, if a user has a *Shared Space Browser* and a *KA artefact annotation tool* simultaneously in use in the KP-Lab Portal, the *Shared Space browser* may highlight the SS into which the KA belongs to. The KP-Lab Portal tools may be either JSR-168 compliant portlets [Abdelnur and Hepper, 2003] (local or remote) or external applications that are not integrated into the system and that do not run on a portal server. The external application tools may appear as a simple link that takes the user to a web page on a server where the tool is installed or to a web page from which the user may download the tool to his own computer. Local portlets are portlets that run on the KP-Lab Portal server and remote portlets are portlets that run on another portal server, yet are displayed on the KP-Lab Portal server. Remote portlets are deployed on the KP-Lab Portal server using the Web Services for Remote Portlets (WSRP) standard [Kropp et al., 2003].

The KP-Lab Portal will be implemented either by using a portal server (such as the Apache Jetspeed-2) and as links to external resources or by using, customizing and expanding an existing Collaborating and Learning Environment (CLE) such as the Sakai Portal (see e.g. http://www.sakaiproject.org/). The external resources which were referred to in conjunction with the usage of a portal server may be, for example, servers where stand-alone software such as Cmap (http://cmap.ihmc.us/) has been installed and where it may be used and web pages from which any arbitrary software may be downloaded.

The overall architecture of the KP-Lab Portal is depicted in Figure 7.

The rest of this section will describe the Portal Level Tools, the Data Integrity Services, the database management systems, i.e. the Knowledge Repository, the Content Repository (or Repositories) and the User Database (DB). All of these are continuously available in the KP-Lab Portal. The rest of the tools and services are only available if needed or requested. They will be described in the subsequent sections.

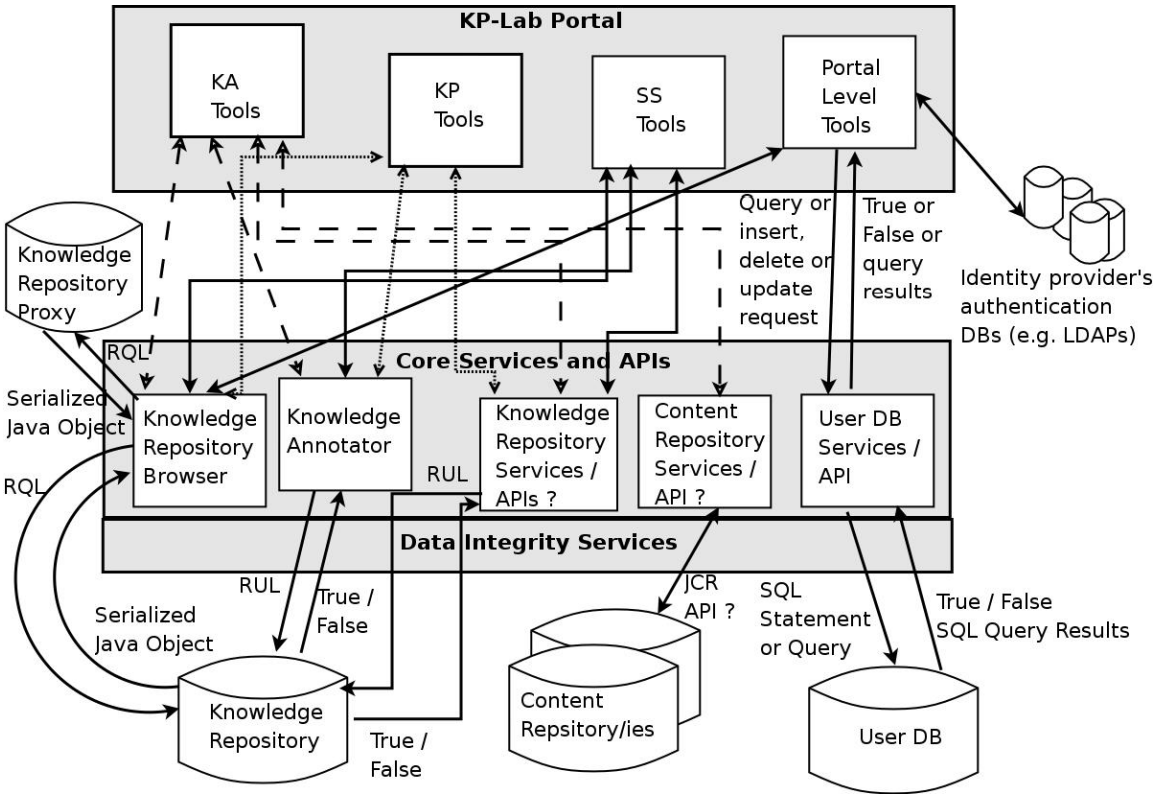

Figure 7 Overall architecture of the KP-Lab Portal. The arrows that leave from the rectangles representing the KA and KP tools are different from the other arrows only to increase the readability of the picture.

The data integrity service ensures that the data stored in the databases is consistent. This can be achieved if a piece of data is stored only once and if all references to it are deleted, if that piece of data is deleted. Also some additional checks might be necessary as illustrated later by the example of the User DB and the SS data stored in the Knowledge Repository. The need for such checks may be expressed by certain constraints that the data must fulfill. We suggest that each database management system i.e. the Knowledge Repository, Content Repository and the User DB is associated with a component that provides functionality that will effectively support integrity of all data in the system. For example, this utility would upon deletion of some KA (or upon request) search the Knowledge Repository, delete all references to this KA and send some notification to the SS that is displaying the KA or references to the KA. The utility would also delete the contents of the KA from the Content Repository. In this manner we do not have to search the Knowledge Repository and delete references manually; resulting in a lower number of RQL queries and RUL updates into the knowledge middleware.

This functionality could be implemented, for example, by providing event sources with methods that allow the developer to register event listeners with them. When an event happens to the source (e.g. a KA is deleted), the source sends a notification of that event to all the listener objects that have registered for that event. A listener object that has registered for the KA deleted event could, for example, be a client displaying a specific shared space to a user. The Knowledge Repository provides a *Registration – Listening Mechanism* (see Section 5.1.2 of the Specification of the SWKM Architecture (V1.0) and Core Services [Christophides et al. 2006]). This mechanism will be used to provide data integrity of data stored in the Knowledge

Repository and to propagate information about changes in data to the Content Repository and to the User DB.

Another example of the data integrity functionality is the change of username and organization of a user. Because the username is the ID of the database record, this operation is broken down into an insert and a delete command. When the delete command is given, a notification is sent to all listeners registered for that event. If the user is the only remaining member of a shared space, then the end user performing the deletion operation should be warned about the fact that there is going to be a trailing shared space with no members left. Two possible actions should be suggested to the end user: 1) Either add another member to the shared space or 2) Delete the shared space. The necessity for providing the user with the warning could be expressed with a database constraint that declares that a shared space must always have at least one member. Otherwise data integrity is violated.

4.2 The Portal Level Tools

The portal level tools are always available to the end users. They provide the following functionalities:

- user authentication (An end user tool. Also called login tool.)
- single-sign-on (SSO) (Not an end user tool.)
- user registration (Not an end user tool.)
- portal statistics (A viewer is provided for the end user.)
- user preferences management (An end user tool.)
- shared space browser (An end user tool.).

These functionalities are described further in the following paragraphs. The *user authentication* function is provided by a federated identity solution such as Shibboleth or Liberty Alliance.

Single-sign-on is provided by the federated identity solution. If needed, the SSO functionality provided by the Portal server (e.g. Jetspeed-2) is coupled into it.

User registration functionality is provided the first time a user authenticates himself into the KP-Lab Portal. This functionality is performed in the background and the end user does not notice it. At registration time, only the information provided by the identity provider is inserted into the User database. The user is invited to add more information about himself into the User database using the User preferences management tool.

Portal statistics are provided, for example, by the portal server (e.g. Jetsepped-2) or by the servlet/JSP page container (e.g. Tomcat). Portal statistics may include information such as portlet specific usage logs or number of users online. The end users will be provided with some statistics information.

The *user preferences management* may be provided by the portal server. The preferences are stored in the User DB using the User DB API. The user preferences may contain any attributes and they are managed using an attribute administration portlet, such as the one provided by the

Jetspeed-2 portal. Even though the attributes may be chosen at will, the Portlet Specification defines a set of attribute names which are recommended to be used. Using these attributes may be handy in the longer run when using WSRP to deploy portlets that are hosted on a remote server, since the use of arbitrary user attributes might cause a mismatch between the attribute names needed by the portlet application and the concrete attribute names that are stored in the Jetspeed-2 User Preferences.

The *shared space browser* functionality is provided at the portal level. The shared space browser may, for example, allow the user to browse all shared spaces or to browse only those shared spaces in which they are involved.

4.2.1 API Specifications for some of the Portal Functionalities

This section provides the API specifications to the Portal Functionalities (or services), except for the single-sign-on, the portal statistics and the user preferences management. These three APIs are excluded because they are most probably provided (and thus also defined) by the Portal server and the federated identity management software. Figure 8 illustrates the APIs of the portal level tools. The APIs are marked with an arrow.

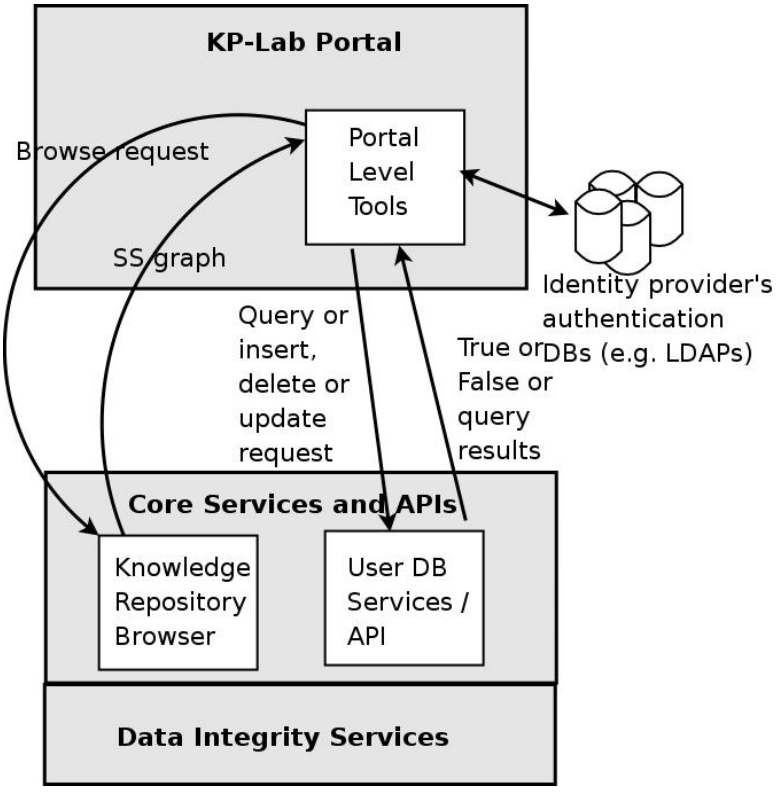

Figure 8:The APIs of the Portal Level Tools.

The notation used in describing the API methods is such that the question mark (?) signifies optionality, i.e. the parameter followed by a question mark is optional. In practice, the methods containing a question mark are overloaded. For those methods that return a Boolean value, true means that the method has succeeded and false that it has failed.

4.2.1.1 Authentication/Login Service

The login service API consists of the following two methods:

- Boolean login(String userName, String password, Organization organization)
- Boolean logout(String username, Organization organization)

An Object belonging to the class Organization must be one of the Strings in the list HY, TKK, EVTEK, HUJI, UTRECHT, etc. The login service is provided by the user's home organization. After the user is logged in, the portal server (e.g. Jetsepeed-2) takes care of session management and of loading the user preferences. When the user logs out, they become an anonymous user of the KP-Lab portal as they were before they logged in.

4.2.1.2 User Registration Service

The API of the user registration service consists of the following two methods:

- Boolean register(String userName, Organization organization, Role role?, String email?, String givenName?, String lastName? etc.)
- Boolean unregister(String username, Organization organization)

An Object belonging to the class Organization must be one of the Strings in the list HY, TKK, EVTEK, HUJI, UTRECHT etc. An Object belonging to the class Role must be one of the Strings in the list student, teacher, customer, secretary, etc. The parameters of the method register() depend on the identity provider and only the first two are mandatory. This is because different identity providers are likely to provide different user attributes. We will provide several overloaded methods, which will allow us to exploit most of the information provided by the identity provider. The user using the user preferences management tool may provide lacking and optional user information.

The user authorization service is implemented at the home organization of each user. The KP-Lab portal is acting as the service provider and it will contact the identity provider (i.e. the home organization of a user) when the user requests login. The first time the user logs into the KP-Lab Portal will serve as the registration of the user as well. After registration, a database record for the user and his preferences is created into the User DB (see Figure 7). In order to provide the User DB a unique key, a concatenation of the userName and of the organization (e.g. liliaEVTEK) is used.

4.2.1.3 Shared Space Browsing Service

The shared space browsing service API consists of the following methods:

- SharedSpaceGraph BrowseSharedSpace(String sharedSpaceUniqueName, Boolean KA, Boolean KP, int radius)
- SharedSpaceGraph BrowseUserSharedSpace(String userName, Organization organization, Boolean KA, Boolean KP)

The first method enables the browsing of arbitrary shared spaces. The center of the graph to be shown is given by the argument sharedSpaceUniqueName and the maximum size of the graph is given by the parameter radius. The Boolean parameters specify whether the KAs and/or KPs related to the shared space are shown or not. The second method enables browsing the shared spaces into which the user belongs to.
An Object of the type SharedSpaceGraph is a serialized Java Object containing the RDF triplets to be visualized. The visualizations envisaged thus far are a graph and a list. An Object belonging to the class Organization must be one of the Strings in the list HY, TKK, EVTEK, HUJI, UTRECHT etc.

4.3 The Core Services

The core services and APIs consist of functionalities that are common to several sets of tools or that are used by the Data integrity services. By observing Figure 7, one may see that the knowledge repository browser, annotator/linker, knowledge repository services/APIs and content repository services/API are accessed by several sets of tools, i.e. the KA, KP and SS tools. At the moment, the user DB services/API is accessed only by the Portal level tools, but placing it under Core services and APIs is required because the User DB has to be accessed also by the Data integrity services. The functionality of each Core Service and API will be explained in the following paragraphs.

4.3.1 Knowledge Browser

This core service is meant to provide two complementary functions: search and visualisation. Both functions concern ontologies and knowledge artefacts. Although, most functionalities are common to these two types of information. However, the following description makes some assumptions:

- access is read only
- knowledge repository is a set of hyper-graphs
- no distinction is made between types of artefacts (e.g. annotations, tasks, processes…)
- an ontology is a namespace that can be transformed into a hyper-graph
- an knowledge artefact is a hyper-graph
- a hyper-graph is a set of nodes and edges

The service interacts with the kernel of the shared space to manage exchange of information with any other internal or external tool. However, it interacts directly with the knowledge repository by submitting RQL queries and retrieving RDF representations. As such, any information concerning individual, groups, tasks, processes etc. will be provided by the shared space. The knowledge will be provided by the knowledge middleware.

User interactions

The user can undertake interactions such as the following:

- look for existing ontologies
- classify ontologies by subject or keyword
- find out relationships between ontologies
- track the evolution of an ontology
- compare two ontologies or versions of ontologies
- discover artefacts related to a given set of ontologies
- find the ontologies describing a given artefact
- find artefacts of an actor (individual, group)
- find artefacts related to a subject or keywords
- track the evolution of an artefact
- view part of a knowledge artefact
- change the point of view into an artefact
- modify the visual appearance
- cluster parts of an artefact
- filter an artefact according to a subset of an ontology
- search for patterns into artefacts
- search artefacts by examples

Functionalities

The user browses the knowledge repository through traditional menu and by clicking directly on diagrams. The retrieval function provides an entry point into the Knowledge Space (i.e. the whole content of the knowledge repository). The tool has at any time a *current artefact*. Such an artefact defines a point of view within the knowledge space. The radius property of the graph represents the depth of the sub-graph to visualise (the number of relations to view starting from the current node). From this point of view, the user can navigate the hyper-graph. This requires an interactive hyper-graph leading to synchronous communication with the knowledge space to avoid uploading large volumes of data (cache mechanisms could be useful).

Basic functions include visualization functions such as:

- display a hyper-graph
- change the point of view,
- change the topology of the graph (collapse/expand nodes, refine the graph, move nodes, reshape edge…),
- change graph perspective,
- modify graph representation.

Other basic functions include:

- presentation of the list of classes and properties making an ontology or an RDF artefact
- transform an RDFS or RDF into a hyper-graph
- associate default visual representation for nodes and relations
- compare 2 RDFS and give a visual representation of the differences
- compare 2 RDF and give a visual representation of the differences
- request RDF (artefacts) with RDFS concept

- request RDFS with a subset of RDFS
- filter RDF hyper-graph on the basis of RDFS elements or on the basis of RDF elements.

The tool will provide two views: an ontology view and an artefact view. The ontology view provides functions (RDFS management) such as

- ontology retrieval from the Knowledge Repository
- ontology relationship view:
    - list of versions of an ontology
    - historic of the construction of an ontology
- ontology view
    - textual view of the ontology
    - graphical view
- compare two ontologies

The artefact view includes the following functions (RDF management):

- artefact retrieval from the Knowledge Repository
- artefact relationship view:
    - list the versions of an artefact
    - historic of the construction of an artefact
    - list of ontologies describing an artefact
    - list of sub-artefacts of an aretfacts
    - list of super-artefact of an artefacts
- artefact view
    - textual view of an artefact
    - graphical view

The tool works in figure 9 depicticting the KPLAB domain model.

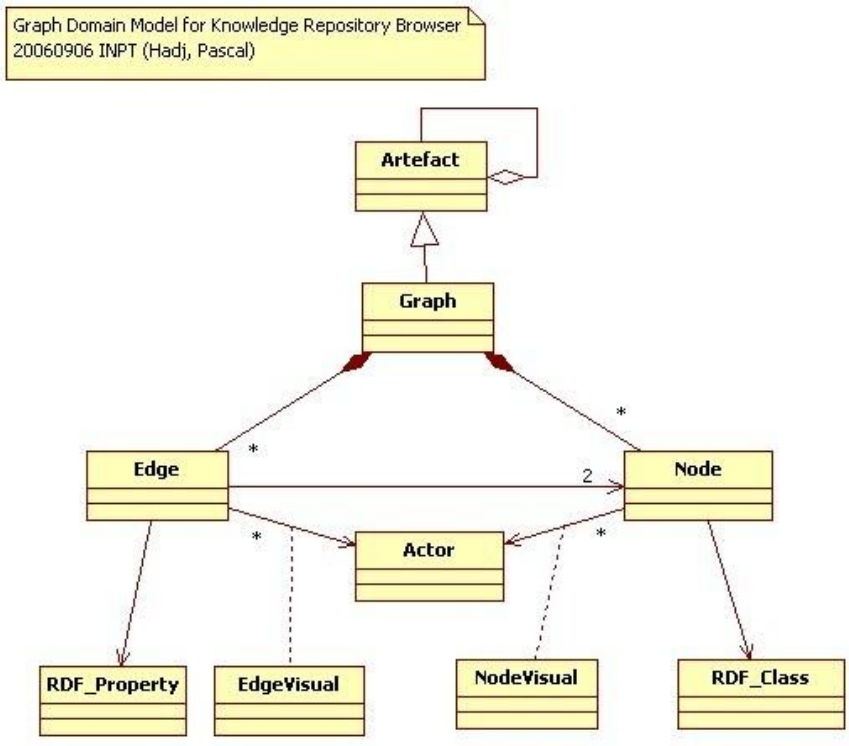

Figure 9: View of the domain model pertinent for diagramming

4.3.2 Knowledge Annotator

The Annotator core services provide methods to annotate every item in the shared space(KAs, KPs and SS)

Methods of the Annotator API:

- boolean *addStatement*(KnowledgeArtefact idKA, int idSS, RDFStatement statement);
- RDFStatement[] *loadStatements*(KnowledgeArtefact idKA, int idSS);
- boolean *editStatement*(KnowledgeArtefact idKA, int idSS, RDFStatement statement);
- boolean *deleteStatement*(KnowledgeArtefact idKA, int idSS, RDFStatement statement);
- RDFSchema[] *loadOntologies*();

The RDFStatement is an object that allows to create RDF statements, ready to be saved into the Knowledge Repository.

RDFSchema is an object that contains information about the ontologies.

4.3.3 Knowledge Repository Services

The Knowledge Repository Services provide the functionalities of forming RQL queries and RUL statements that can be passed on to the Knowledge Repository or the Knowledge Repository Proxy and of communicating with the Data Integrity Services. These services are used by the Shared Space Management Tools in order to update Shared Space specific information using the Information, User and Tool Management functionalities and by the Knowledge Annotator and Knowledge Browser. The knowledge repository services also manages the Knowledge Repository Proxy.

The Knowledge Repository Proxy is used to make the use of the Knowledge Repository faster. For example, a user may first use the Knowledge Browser to upload a graph from the Knowledge Repository. This graph is stored into the Knowledge Proxy and it is displayed to the user. After that the user queries the graph. The scope of the query is only the small portion of the whole graph that is stored in the Knowledge Repository Proxy and the query is naturally passed only to the Proxy and not to the Knowledge Repository. The knowledge repository services take care that the query is always first passed to the proxy, and only if needed to the Knowledge Repository. The Knowledge Repository Services also take care of removing old graphs from the proxy when it becomes full.

Another example of the usage of the proxy is simultaneous collaborative editing of a graph. The temporary changes are stored in the proxy and only the final version is stored into RDFSuite. The Knowledge Browser shows on the screens of the users the proxy's version of the graph.

4.3.4 Content Repository Services

The Content Repository Services consist in services and APIs between the Knowledge Artefact Tool and the JSR-170 compliant Content Repository as well as of communicating with the data integrity services. All content stored in the Content Repository also contains metadata, and is thus part of a Knowledge Artefact. The Content Repository Services provide each piece of content with a URI that will be used when a reference to the content is stored in the Knowledge Repository.

4.3.5 User DB Services

The User DB Services consist in forming an abstraction between the User DB and the tools that use it, i.e. the Shared Space and Portal Level Tools as well as of taking care of the communication with the Data Integrity Services (see Figure 9).

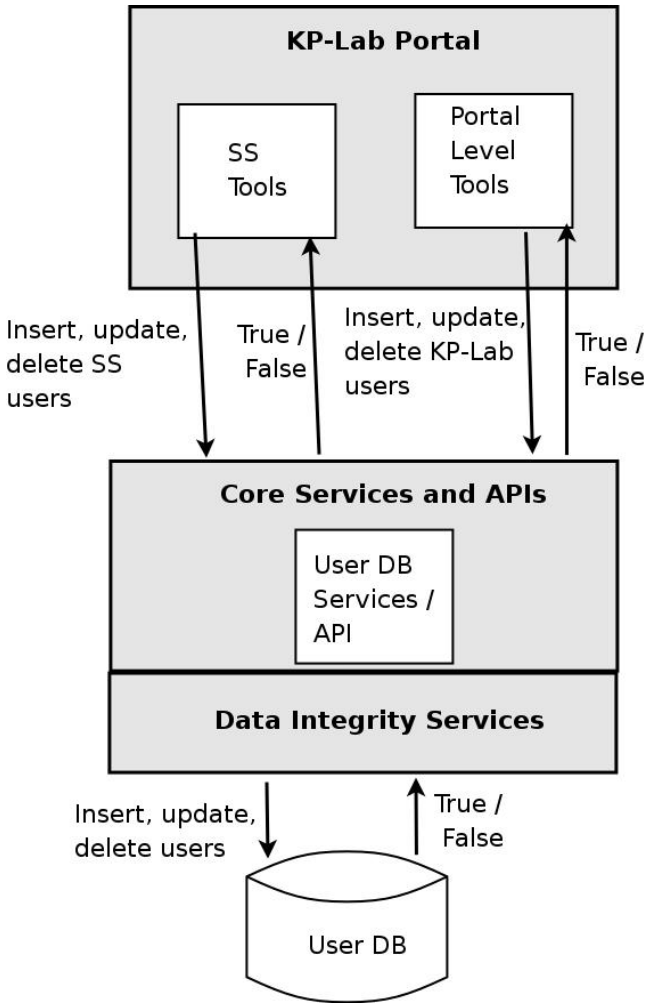

Figure 10:The APIs of the User DB Services. The APIs are illustrated by the arrows.

4.4 The Data Tier

The data tier consists of three different database management systems: the Knowledge Repository, the Content Repository and the User DB. The Knowledge Repository is provided by the Semantic Web Knowledge Middleware (SWKM) and it is developed in WP5. It will contain data about KAs, KPs, SSs and users. The Knowledge Repository is used to store only formal knowledge, i.e. knowledge that is expressed by RDF (Resource Description Framework) triplets that uses a vocabulary (or ontology) expressed by RDF Schema (RDF-S). At a later stage, ontologies expressed by OWL (Web Ontology Language) may be considered to be used instead of RDF-S. An RDF triplet consists of three elements: the resource, the property and the property value. The resource has to be identified by an URI. The database management system for storing informal content is a JSR-168 (i.e. the Java Portlet specification) compliant Java Content Repository (JCR). In this document, it is called the Content Repository. The third type of database to be used is the User DB. The database management system for storing this data is a relational database that is used only by the Portal Tools. For the purpose of user authentication, the user's home institution's DBs will be used.

They may be LDAP DBs. The implementation of these DBs is up to each institution, and it will not be discussed further in this document.

4.5 Knowledge Artefact Tools

4.5.1 Technologies

The KA tools will be implemented as JSR-168 portlets and deployed on the same portal / portlet container together with the rest of Shared Space tools.

JSF and JSP (or facelets) will be used to implement the portlets.

4.5.2 Content Repository

The Content Repository can be implemented in many ways: relational database, object-oriented database, files in a directory structure, custom-made content storage facility or JSR-168 compliant Java Content Repository (JCR).

As it was already mentioned (see [WP6-5]) it seems that using JSR-168 compliant repository will be a good match to the requirements of the Shared Space Content Repository:

- hierarchical n-ary tree model
- metadata is supported via properties
- versioning
- locking
- the nodes can be made referenceable via UUIDs
- notification on changes
- searching witch XPath
- open source implementations are available (Jackrabbit, eXo JCR, Jeiceira)

Figure 11 shows a rough idea of the topology of JCR based Shared Space content repository. A special namespace – *kpl* – is used to distinguish the KP-Lab shared space specific *items*.

The nodes which represent parts of knowledge artefacts are denoted as *kpl:ka*[*i*]. The KA nodes are of base type *nt*:*unstructured* (in some cases *nt*:*folder* will be useful as well) with the addition of the following *mixins*:

- *mix:versionabe* – to provide for versioning support
- *mix:referenceable* – to instruct the JCR to auto-generate UUIDs
- *mix:lockable* – if locking is required while the content is checked out for modifications.

Figure 11 also shows the possibility to manage annotations using JCR. They correspond to a comment on the content and are represented as *properties* associated to the KA *nodes*.

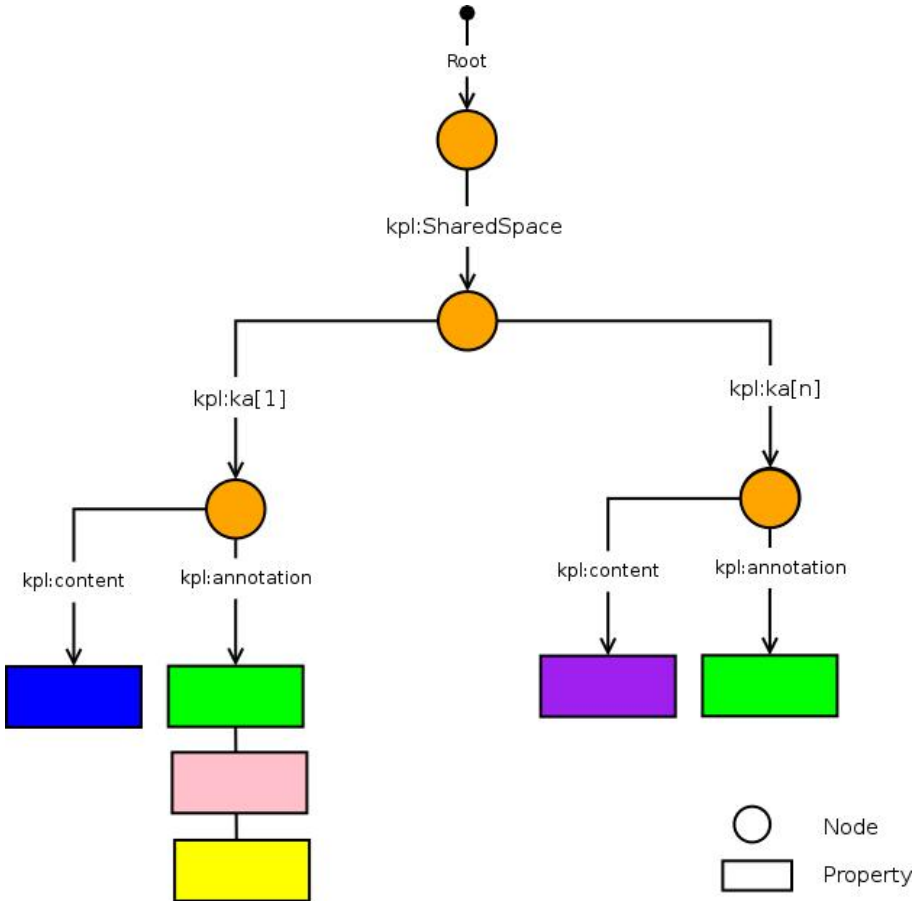

Figure 11: Conceptual diagram of the Content Repository

4.5.3 Knowledge Artefact Manager

Methods of the API:

- KnowledgeArtefact *createKA*(String name, int idSS);
- boolean *deleteKA* (KnowledgeArtefact ka);
- KnowledgeArtefact *searchKA*(KnowledgeArtefact ka, String static Metadata);
- boolean *saveKA* (KnowledgeArtefact ka, String content, String staticMetadata);
- boolean *editContent*(KnowledgeArtefact ka, String content);
- boolean *notifySS*(KnowledgeArtefact ka);
- void *annotate* ();

The methods *createKA*, *deleteKA*, *searchKA*, *editContent* and *annotate* correspond to functionalities that are accessible from the Shared Space (with right click on a KA). The method *annotate* is common to any item of the Shared Space and it opens the Annotation Tool.

The method *notifySS* calls the Shared Space Management Tool *updateContent* method to refresh information about a KA.

4.5.4 KA Mapper

The role of the KA mapper is to associate a KA with a specific region of a 2D image.

As a future extension the 2D image can be replaced with a more generic coordinate space – x,y,z,t (3D + time).

Here we will adopt the approach of storing the image and the mapping in the JSR-168 content repository. The background image is a node in repository with the following specifications (Figure 12):

node name: *bgimage1.png*
primary type: *nt:file*
mixins: *mix:versionable, mix:referenceable*
properties: *jcr:UUID* - referenced by the mapped KAs
child node definitions
name: *jcr:content*
properties: *jcr:data* - the binary image is stored here

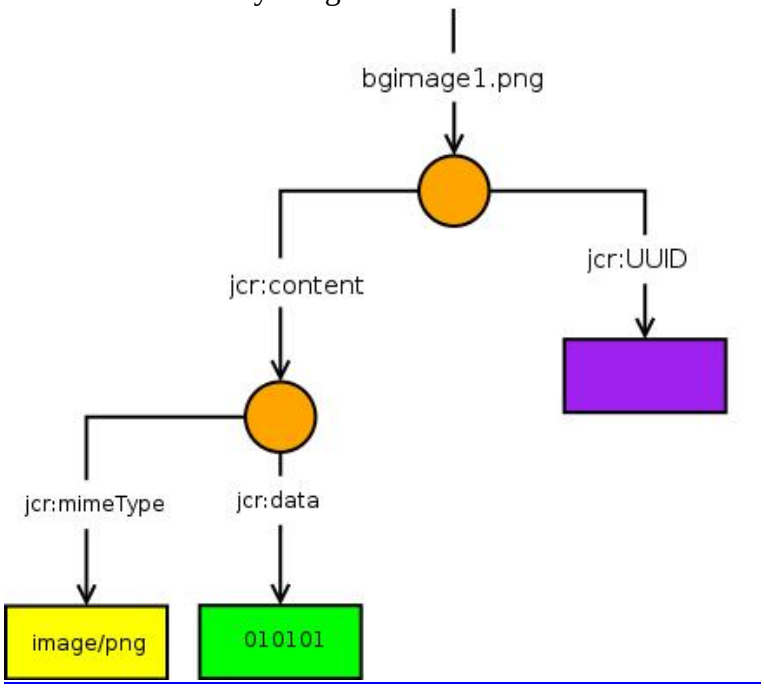

Figure 12: Background image node.

The mapping between the KA and the background image will be represented as a property of the KA node. The value of this property will include the UUID of the image node and the coordinates of the KA on the image (x, y) or (x, y, z, t) at later stage.

The proposed mapping mechanism allow for multiple KAs to be mapped on the same background image, if deemed necessary.

A natural evolution of the idea of mapping to images would be to integrate Google Map services into the mapper tool for cases when the artefacts need to be geographically correlated.

4.6 Knowledge Process Tools

In this section we describe Knowledge process tools architecture and its connection to other parts of the Shared space. The overall architecture of the Knowledge Process Tools is depicted in Figure 13.

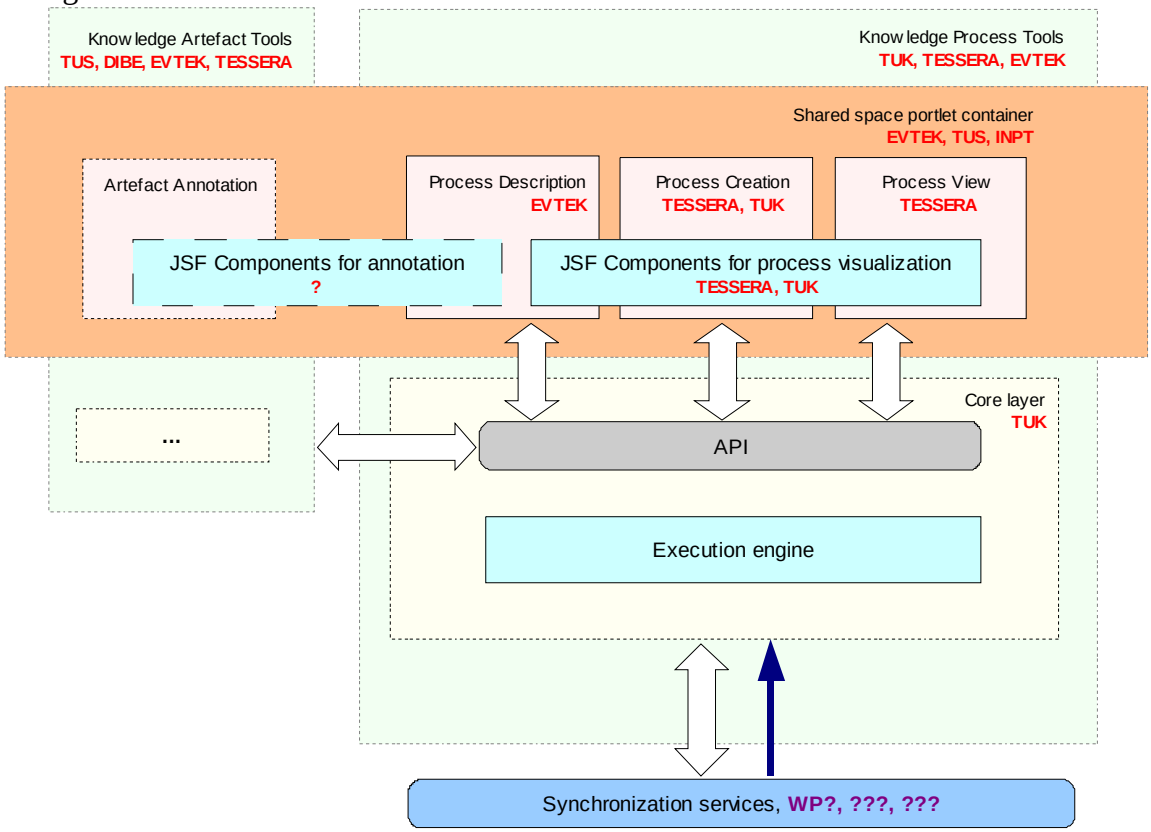

Figure 13: Overall architecture of the Knowledge Process Tools

Core layer:

Core layer will be developed mainly by TUK. Role of the core layer is to:

1. Provide high level representation of processes and operations with them;
2. Ensure that knowledge process is valid, tasks are consistent and dependencies are also valid;
3. Manage execution state;
4. Manage storing of knowledge processes into Knowledge repository (see also process ontology at the end).

Core layer will use Object Model provided by SWKM (WP5) in order to have fast access to the knowledge model. SWKM shall provide a client-side object model.

Knowledge processes performed within the KP-Lab tools using Shared space mainly, will also be defined by means of process ontology, presented on the following Figure 14. This image is visualization of suggested ontology made in ontology editor, called Protégé 2000. This is not an UML diagram.

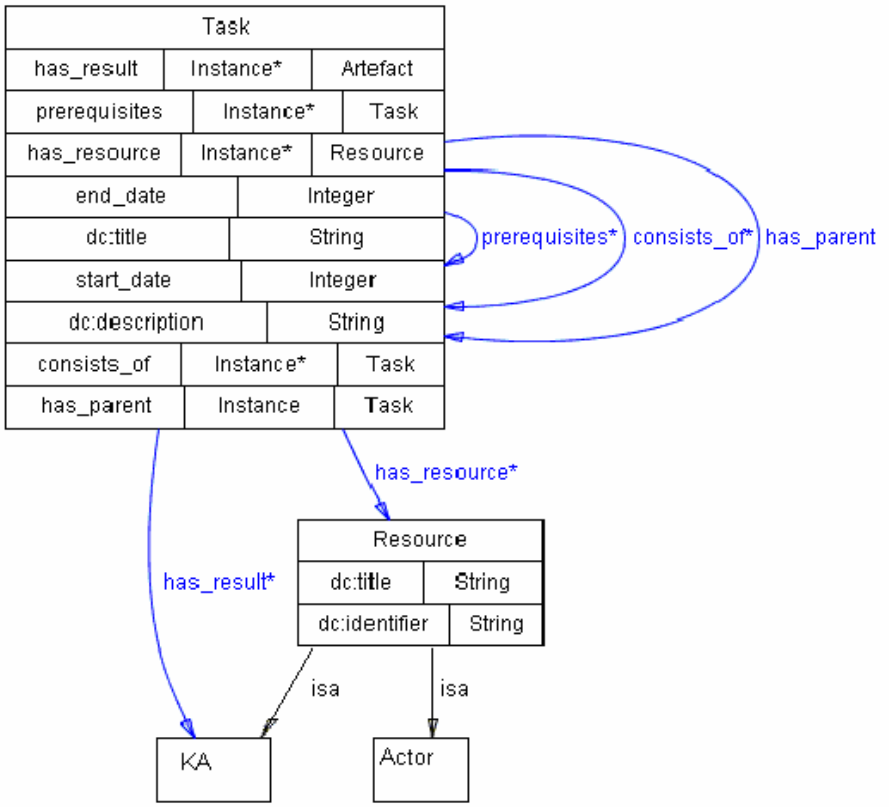

Figure 14: Process ontology

Process ontology legend:

- Prerequisites – tasks, which must be completed before execution of actual task can be started.
- Consists_of – link to subtasks, which belong to actual task
- Has_parent – link to task, which contains actual (sub)task (ancestor). This slot is inverse to consists_of, to allow bidirectional traversal.
- Resource – this is a term borrowed from project management and means everything what is needed in order to complete successfully the task (e.g. relevant knowledge artefacts or actors).

This is preliminary representation of a generic process in form of ontology. Each process will be stored in this form into the knowledge repository.

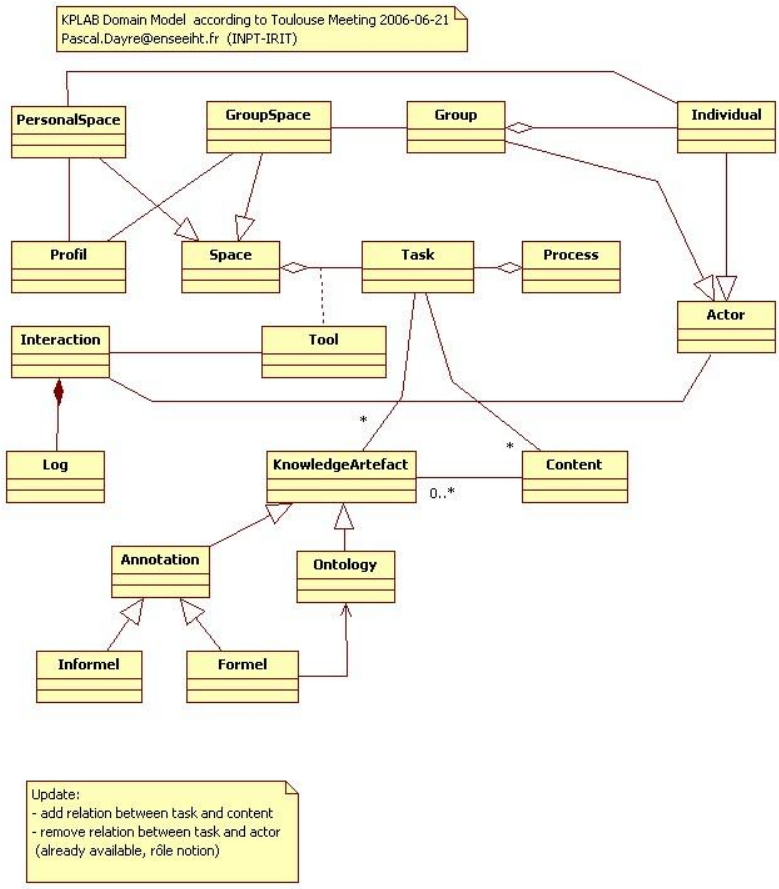

Figure 15: KP-Lab Domain model

The process ontology corresponds also with the KP-Lab domain model (see Figure 15), but we are not using the concept "Content" here, because there will not be a clean content in the KP-Lab. Each piece of data (e.g. from external repository) will be immediately described with at least some basic metadata and as such becomes a knowledge artefact.

Execution engine manages state of executed processes represented as a set of instances of the KP-Lab process ontology. It also manages validity, dependency and consistency of knowledge processes.

API layer provides high-level interface for working with knowledge processes. It will provide JavaBeans representation of tasks and processes and will also provide factory class for creating and editing knowledge process and methods for saving/loading of a process from database. API layer is of course fairly open thing and we can provide other representation of knowledge process or modify API depending on our partners' needs.
API layer will also maintain internal cache of processes in order to postpone updates to SWKM until user finishes updating process or its components. This is of course optional and will lower amount of communication with the knowledge database.

Artefact view maintained by Knowledge artefact tool sometimes also displays tasks and subtasks of a knowledge process. Because of this, it will probably also have to use API layer to query for tasks and its parameters.

Presentation layer:
Presentation layer is mainly developed by TESSERA. This layer will tightly cooperate with shared space component that will serve as container for knowledge process tools. JSF Components will be implemented for visualizing the KP tools (Create, Delete, View/Update, and Annotate) and call/invoke the necessary API methods/functions needed from the core layer.

These components will be used in all three front-end portlets (process view, creation and annotation). This way we ensure consistent look and usability across every knowledge process tool. These components will be generic enough to allow them to be used both for viewing, creation and modification of process or tasks.

JSF Components for annotation/description will be developed together with partners from Knowledge artefact part. This will ensure that annotation of Knowledge artefact and annotation/description of Knowledge process will have a consistent look.

In **Process View** portlet, user will be able to create or modify structure of process, create and modify tasks and subtasks, modify their description and assign knowledge artefacts related to specific task.

Process description portlet should look and behave similarly to description/annotation portlet in Knowledge artefact tool and is of course used for describing tasks of a knowledge process.

4.7 Shared Space Management Tools

Figure 16 presents the APIs of the KP-Lab shared space management tools.

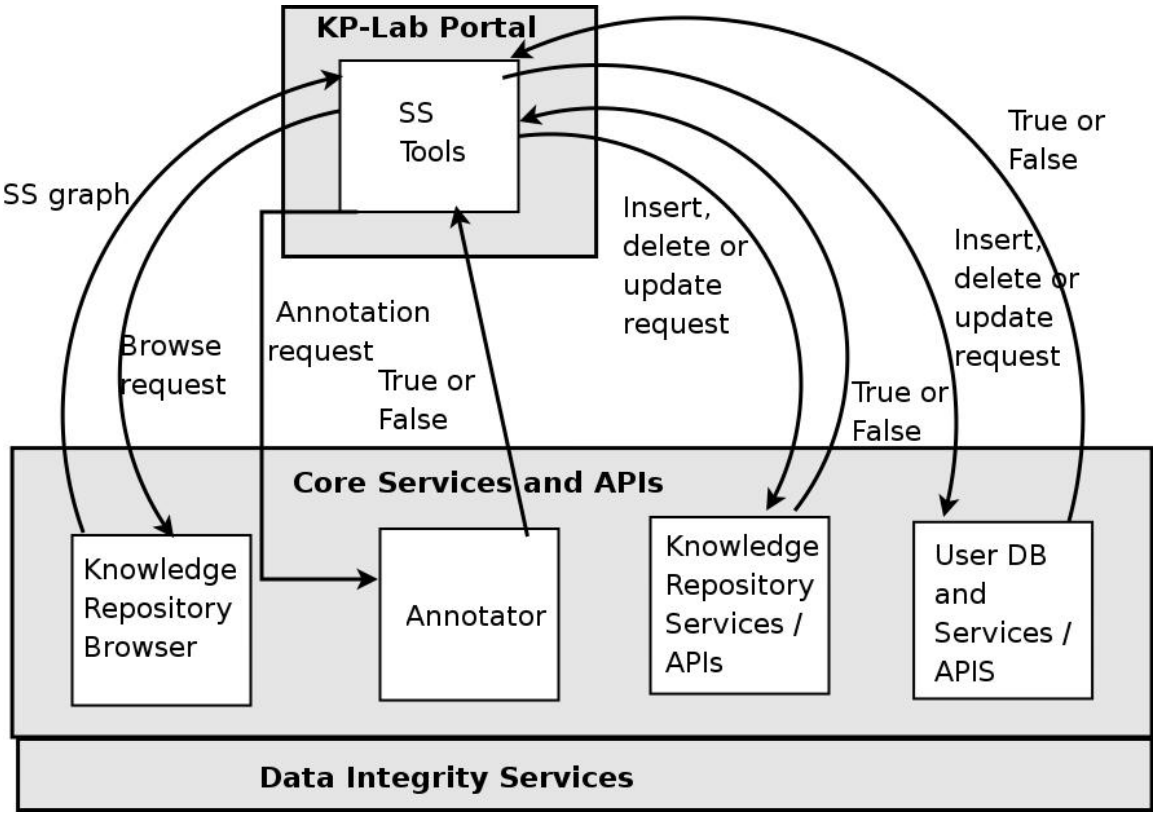

Figure 16: The APIs of the Shared Space Management Tools are illustrated by the arrows.

In the following subsections, the API for the Shared space tools will be specified.
In order to view a shared space, the KP-Lab Portal Tool Shared Space browser is used. This tool and its API was described in Section 4.1. The notation used in describing the API methods is the same as in Section 4.1, i.e. such that the question mark (?) signifies optionality. For those methods that return a Boolean value, true means that the method has succeeded and false that it has failed. All of the methods belong to the class SharedSpace and must be applied to an instance of an Object of the type SharedSpace. This avoids us from having to specify the ID of the Shared space as a parameter to all of the methods.

4.7.1 Shared Space Constructor(Creator)/Deletor

Methods of the API:

- SharedSpace(List<String> managerMember, String parentSharedSpaceUniqueName?)
- Boolean delete()

The first method is the constructor method and the second method is a destructor method for the Shared space. The constructor takes care of the uniqueness of the Shared Space ID. A Factory class might also be provided instead of this simple constructor.

4.7.2 Shared Space Annotator

The Shared Space Annotator calls the Knowledge Annotator through an API that could approximately look as follows:

- Boolean annotateSS(String SharedSpaceUniqueName, String toAddName, Type type)

An Object of the type Type may be C (Content), KA (Knowledge Artefact), KP (Knowledge Process) or SS (Shared Space). If an Object of the type Content is added to the shared space, the Knowledge Annotator calls the Content Repository Services and a file import or creation functionality is called and the content is stored into the Content Repository. The new content is a Knowledge Artefact because it is composed of the content (the imported file) and of metadata (the id of the shared space into which it was added, among others). If a KA, KP or SS is added to the shared space, the functionality may result in either creating a copy or in creating a reference. KAs in personal spaces are always copies and KAs in a collective space may be either copies or references. However, a collective space may not contain a reference to a KA in a personal space. In this case, a copy has to be made [D5.1].

Dependencies to Core Services and APIs:

- There is a dependency upon the Knowledge repository browser, such that: in order to produce the GUI in which the user may define into which Shared space he wishes to add something. If the user is copying or linking something from another shared space, then the Knowledge repository browser service is needed in this phase also.

4.7.3 Information Management

- Boolean updateInformation(List<String> managerMember?, List<String> parentSharedSpaceUniqueName?, String spaceDescription?, String level?, DateFormat startDate?, DateFormat endDate?, AccessDefaults access?)

An Object of the type AccessDefaults may be for example *private* (only members of the space can read, add, modify, move, organize, delete... the content), *protected* (only members can add, modify, delete the content, but a visitor can read it) or *public* (everybody can do everything with the content). The managerMember List may not be NULL because a shared space must have at least one managerMember. If the parentSharedSpaceUniqueName List is empty, the default parent shared space – called the root shared space - will be set as the parent of the shared space. This tool only sets default access rights to different user groups or roles (in the case of the example, the groups were: member, not member and anonymous). In order to set user specific access rights, the User Management tool (see Section 4.7.4) is used.

This method overwrites the information that is specified by the parameters and leaves untouched those that are not specified.

4.7.4 User Management

- Boolean AddUser(String username, Organization organization)
- Boolean updateUserRights(String userName, Organization organization, UserRights rights, ContentType type?, ID id?)
- Boolean removeUser(String username, Organization organization)

An Object of the type UserRights is a combination of different rights: e.g. read, add, delete, execute, comment. The Objects of the type Organization and ContentType are defined as above in the other methods. ID is a unique identifier referring to a KA or KP. The user rights may thus be set separately for a specific content type, e.g. all KAs or for a specific KA.

4.7.5 Tool Management

- Boolean allowTool(String toolUniqueName, ToolType type)
- Boolean forbidTool(String toolUniqueName, ToolType type)

An Object of the type ToolType may be either portlet or link. If it is a portlet, then the tool will be added as a portlet to the shared space. If it is a link, it will be added as a link to a web page containing the tool or providing a download facility for the tool. From the implementation point of view, the portlet containing the tool will be added to the portal server.

5 Conclusions

5.1 Problems Encountered

The requirements process has turned out to be even more challenging than expected. The requirements gathering through the co-evolution process and the work in design teams have been challenging and somewhat delayed from the initial project plan. The problems encountered have been documented in [D2.1].

The pedagogical scenarios focus on high level learning activities and do not lend themselves to specific technical requirements required in the software design. Another problem with the pedagogical and professional scenarios is their heterogeneity, which makes it difficult to establish a shared understanding of functionality and features needed. In order to avoid unacceptable delays, focused design groups with limited number of partners were established. This allowed us to define more specific requirements on the expense that fewer scenarios were covered.

5.2 Next steps

The decision on field trials for spring 2006 will be made by September 15, 2006. The user stories describing these scenarios will be reviewed and finalised in collaboration between the pedagogical partners of WP8 and the technical partners of WP6. This will allow us to freeze the functional requirements for the release 1.

WP6 partners have already started the detailed planning of the design and implementation of the software release 1. This will be finalised by September 15, 2006 with a detailed division of labour and milestones for the development and implementation. Two intermediate integration releases have been planned for mid-November and mid-December.

The release for first field trials will be available at the end of January 2007. WP6 partners will provide necessary training and support for the users during the field trials.

6 Bibliography

[Abdelnur and Hepper 2003] Alejandro Abdelnur and Stefan Hepper: Java Portlet Specification, Version 1.0. October 7, 2003.

[Adams 2003] Adams, M., Edmond, D., Hofstede, A.H.M.: The application of Activity Theory to Dynamic Workflow Adaptation Issues. Proc. of the 7th Pacific Asia Conference on Information Systems, July 2003, Adelaide, South Australia [AHM06] H. Ahola and H. Markkanen. T4.3.1 Identity, Authorisation Management, and Security Modeling: Survey on Shibboleth approach. KP-Lab internal deliverable for T4.3, April 2006.

[Barik 2005] Titus Barik, Introducing the Java Content Repository API http://www-128.ibm.com/developerworks/java/library/j-jcr/index.html

[D2.1] KP-Lab project deliverable 2.1; http://www.kp-lab.org/intranet/work-packages/wp2/deliverable-2.1/)

[D3.1]. Kp-Lab project Deliverable 3.1; http://www.kp-lab.org/intranet/work-packages/wp3/deliverables/d3-1/).

[D5.1] Christophides, V., Kotzinos, D., Smrz, P., Furdik, K., et al.: Specification of the SWKM Architecture (V1.0) and Core Services. KP-Lab Deliverable D5.1, July 2006.

[D8.1] KP-Lab project deliverable D8.1: "Scenarios and User Requirements for KP-Labs in Education", August 2006, http://www.kp-lab.org/intranet/work-packages/wp8/deliverable-8.1/deliverable8-1.doc/view

[DT-1] Pedagogical and Professional Scenarios. http://www.kp-lab.org/intranet/design-teams/pedagogical-and-professional-scenarios/

[DT-2] Multiprofessional teamwork and use of dynamic workflow tools. http://www.kp-lab.org/intranet/design-teams/dt15-processes-and-artefacts/pedagogical-and-professional-scenarios-of-dt-15/multiprofessional-teamwork-and-use-of-dynamic-workflow-tool/ [WIKI-1] Design Principles. http://kplab.evtek.fi:8080/wiki/Wiki.jsp?page=Design Principles

[Hakkarainen 2006a] Hakkarainen, K.: Scientific challenges of KP-Lab. Working papers of the KP-Lab project. Available at the KP-Lab intra: http://www.kp-lab.org/intranet/work-packages/wp3/background-materials/hakkarainen-scientific-challenges-of-kp-lab.doc/view

[Hakkarainen 2006b] Hakkarainen, K.: Design challenges of KP-Lab. Working papers of the KP-Lab project. Availabe at the KP-Lab intra: http://www.kp-lab.org/intranet/design-teams/hakkarainen-design-challenges-of-kp-lab.doc/view

[Kropp et al., 2003] Alan Kropp, Carsten Leue, Rich Thompson (editors): Web Services for Remote Portlets Specification. Approved as an OASIS Standard. August 2003.

Lawrence Erlbaum. Retrieved November 19, 2004, retrieved from *http://www.helsinki.fi/ktl/opiskelu/tt_mo_2006/PaavolaLipponenHakkarainen2004.pdf*

[Marshall 1998] Catherine C Marshall, Toward an ecology of hypertext annotation, Proceedings of the 9th ACM conference on Hypertext and Hypermedia 1998, http://www.csdl.tamu.edu/~marshall/ht98-final.pdf

[Orenl 2006] Eyal Oren1 et al., Annotation and Navigation in Semantic Wikis, http://eyaloren.org/pubs/semwiki2006.pdf

[Paavola 2006] Paavola S., Hakkarainen, K.: "Trialogical" Processes of Mediation through Conceptual Artefacts. Technical Report for the KP-Lab consortium, University of Helsinky, Finland, 2006

[Paavola et al 2004] Paavola, S., Lipponen, L., & Hakkarainen, K. (2002). Epistemological foundations for CSCL: A comparison of three models of innovative knowledge communities. In G. Stahl (Ed.), *Computer supported collaborative learning: Foundations for a CSCL community: Proceedings of the Computer Supported Collaborative Learning 2002 Conference* (pp. 24–32). Hillsdale, NJ:

Resource Description Framework (RDF). Concepts and Abstract Syntax. W3C Recommendation, 10 February 2004.

[SHI06] About Shibboleth. http://shibboleth.internet2.edu/about.html

[TechWP-1] Technological WP. UML modelization of KP-Lab Concepts. http://www.kp-lab.org/intranet/work-packages/technical-work-packages/meetings/toulouse-21-6.2006/wp4-uml-modelization-of-kp-lab-concepts/

[WIKI-2] User Stories. http://kplab.evtek.fi:8080/wiki/Wiki.jsp?page=CategoryUserStories

[WIKI-3] Use case: create a shared space. http://kplab.evtek.fi:8080/wiki/Wiki.jsp?page=UseCaseCreateSharedSpace

[WIKI-4] Use case: Login http://kplab.evtek.fi:8080/wiki/Wiki.jsp?page=UseCaseLogin

[WIKI-5] Use Case: Manage Existing Shared Space. http://kplab.evtek.fi:8080/wiki/Wiki.jsp?page=UseCaseManageExistingSharedSpace

[WP6-1] Shared space demo links. http://www.kp-lab.org/intranet/work-packages/wp6/shared-space-demo/shared-space-demo-links/

[WP6-2] Mock-up requirements and design – Case: Project course at EVTEK v0.3 http://www.kp-lab.org/intranet/work-packages/wp6/shared-space-demo/mock-up-

requirements-and-design/

[WP6-2] Knowledge Process Tool Architecture http://www.kp-lab.org/intranet/work-packages/wp6/architecture-of-the-m12-prototype/t65_arch_02.zip/view

[WP6-3] Knowledge artefact management - Technology survey http://www.kp-lab.org/intranet/work-packages/wp6/t6-4-shared-space/knowledge-artefact-management-tecnology-survey/

[WP6-5] Notes from WP6 meeting in Toulouse, http://www.kp-lab.org/intranet/work-packages/wp6/coordination/meetings/wp6meetingtoulouse-3.ppt/view