WASABIBEANS
Web Application Services and Business Integration

Jonas Schulte
Heinz-Nixdorf-Institute, University of Paderborn, Fuerstenallee 11, 33102 Paderborn, Germany

Keywords: Cooperation, Framework, Integration, Knowledge, Management, Services, SOA, WasabiBeans.

Abstract: Albert Einstein, a German physicist borne in 1879, said: *"Progress requires exchange of knowledge"*. This sentence is more relevant than ever, since projects in economy and research become increasingly complex. Due to the information flood and the growing complexity scientists all over the world do research on knowledge management and organization. This article outlines the particular position of CSCW systems for knowledge organization, since involving people in the process of data structuring and organization is crucial to support (work) process and information retrieval. Furthermore, a framework will be presented to demonstrate some innovative concepts for collaborative knowledge organization and work. The focus is on presenting the powerful authentication and authorization infrastructure and means of flexible user-driven repository integration to build up complex knowledge networks.

1 INTRODUCTION

Business data of a company, latest research results of an university or learning content in education – these are just some of many possible examples to illustrate that knowledge is the key factor in modern society. Knowledge is created by people, managed and shared. One contribution of computer science to support these knowledge processes is the research on *computer supported cooperative work (CSCW)* and *knowledge management (KM)*. Both, CSCW and KM are very interdisciplinary and novel fields of research. CSCW has been substantially shaped by Paul Cashman and Irene Greif through the first conference on CSCW in 1986 (Greif, 1988). In the same year the term knowledge management was mentioned for the first time by Karl Wiig (Liebowitz, 1999).

Collaborative work environments are supposed to support the coordination as well as the teamwork of people in different places. This is mandatory to work efficient on common projects in spatially divided teams. In (Hampel, 2001) basic (media) functions for collaborative work environments are given. Furthermore, the author presents a basic structure to establish work environments in the digital media: the *virtual knowledge spaces (VKS)*. Using the metaphor of a space in which knowledge is stored, structured, and cooperatively generated, virtual knowledge spaces are not only an implementation of the basic functions, but also an intuitive understanding and use of the collaborative work environment by the user. Hence, virtual knowledge spaces are a central concept for building collaborative work environments for everyday use.

The concept of virtual knowledge spaces is most helpful to the organization of knowledge in distributed environments and whenever knowledge exchange takes place in asynchronous work processes. Since, asynchronous work processes can effectively be supported by a CSCW system, it is important to design flexible cooperation support systems that address both collaboration among people and knowledge management. When considering these two research fields, it is important to distinguish between data and knowledge. Data does not make any statement about its interpretation, but knowledge is the ability to interpret the data correct (Dengel, 1994). Justin Hibbard gave the following definition of KM in 1997: *"Knowledge Management is the process of capturing a company's collective expertise wherever it resides – in databases, on paper, or in people's heads – and distributing it to wherever it can help produce the biggest payoff."* (Hibbard, 1997).

However, classic CSCW focus on working with data instead of knowledge. The intention of the *WasaibBeans* framework, presented in this article, is to support collaborative knowledge and information organization in distributed environments. This is relevant to the fact that people are knowledge carriers

Schulte J..
WASABIBEANS - Web Application Services and Business Integration.
DOI: 10.5220/0003509903480357
In *Proceedings of the 13th International Conference on Enterprise Information Systems* (ICEIS-2011), pages 348-357
ISBN: 978-989-8425-56-0

and new knowledge is mainly produced in team work. WasabiBeans is an implementation of the concept of virtual knowledge spaces and bases on latest Java EE[1] technologies. Furthermore, the framework considers the deep influences of the Web 2.0 on CSCW. WasabiBeans does not offer a complete cooperative work environment, instead it provides functions to work with virtual knowledge spaces. As a *service-oriented architecture (SOA)* with a micro-kernel, WasabiBeans is highly expendable and can be customized to different fields of application. Thereby, knowledge processes and especially the externalization of knowledge (see Figure 1) can be facilitated. Problematic is the transformation of explicit to tacit knowledge, since tacit knowledge can not be digitalized and made persistent.

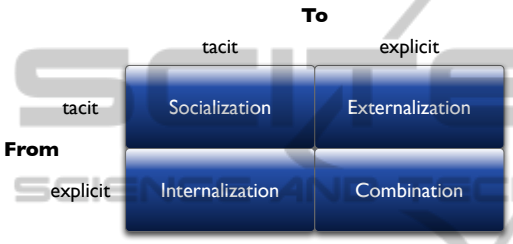

Figure 1: Knowledge Processes by Nonaka and Takeuchi.

To prevent this transformation of knowledge computer science particularly develops solutions to support the externalization of knowledge. Externalization means that the knowledge is extracted from the peoples' head and permanently stored in databases, knowledge management systems or the like. The "destruction" of explicit knowledge occurs when automatizing (work) processes. For example step-by-step instructions may be unnecessary after an automatization. The intention of the WasabiBeans framework is to integrate various systems and applications to embed efficient KM into existing workflows.

In practice, there are numerous systems for knowledge management. Often these systems are monolithic and require a fundamentally restructuring of the business processes and they are not optimized for collaboration and team work. The WasabiBeans framework tries to fill the gap between cooperation support and knowledge organization by providing functions for cooperation support as well as solutions for flexible integration of "arbitrary" (knowledge) databases and repositories.

This article is structures as follows. In Section 2 the concept of virtual knowledge spaces is introduced. Based on this concept the architecture of the WasabiBeans framework is described in section 3. To discuss the framework more detailed section 4 will focus on the authentication and authorization infrastructure of WasabiBeans and section 5 explains WasabiPipes. WasabiPipes is a *pipes and filter (PaF)* architecture to connect different repositories and web applications with the framework. This is very important for cooperative work, since it allows the integration of all necessary knowledge and data repositories. Finally, the results are summarized and discussed in section 6.

[1]Java Enterprise Edition, a widely used platform for server programming in the Java programming language. Online available: http://www.oracle.com/technetwork/java/javaee/overview/index.html

2 KNOWLEDGE SPACES

Virtual knowledge spaces are a key concept to create dynamic and flexible cooperative work and learning environments. It allows to structure the data unrestrictedly and provides means for generating multiple views on the same data collection (see section 2.2). Thus, it is possible to adjust the environment exactly to the users' needs, without having redundancy of the underlying data. The concept of virtual knowledge spaces has been proven for several years and in various fields of application (Eßmann et al., 2006). The following section 2.1 describes the class hierarchy for setting up virtual knowledge spaces. After that we show how to create different views of a data collection (section 2.2).

2.1 Data Model of Knowledge Spaces

Knowledge spaces are our concept for the representation and the structuring of information within a cooperative work process. This the concept has been applied to various fields of application and is well-proven due to almost ten years experience of software development in the scope of CSCW.(Hampel and Keil-Slawik, 2001)

The basic object structure of virtual knowledge spaces is quite easy. We distinguish between the following objects: `Room`[2], `Container`, `Document`, `Attribute`, `Link`, `Group` and `User`. A `Room` is the representation of a real room, whereas `Container` can be compared with a folder. The intention behind the `Room`-object is having an object, that stands for a place where uses can meet each other and cooperate with each other. Furthermore spaces are intended to assure awareness. Awareness is indispensable for successful CSCW applications (Licht et al., 2003; Prinz and Gross, 2001). Besides, the `Container`-object is

[2]In the object model a `Room` stands for a virtual knowledge space.

a lightweight construct to organize information and correlate data. Objects of the type `Document` are used as a wrapper around the real content (e.g. a picture, a PDF file, ...). There is a fundamental difference between on the one hand storing content and linking it with additional information and on the other hand *combined data object* that wrap content, annotations, comments, access rights, and further information. The first option links different data to each other, whereas the second and preferred option builds up a *common space of action*. Working with combined data object might have different advantages. For instance, when creating versions of an object it is a lot easier to generate a coherent version object of a combined data object instead of creating equivalent versions of all related objects that match the combined data object. The data model of WasabiBeans allows free attributes at any type of object, thus the model is highly customizable to the requirements of the current application area. `Group`-, `User`- and `Link`-objects are self-explanatory. It is mentionable, that we use `Group`-objects not only for grouping users, but also to describe roles. The concept of virtual knowledge spaces allows to group and structure combined data objects on a higher level. Our model allows free attributes at any type of object; in this way users keep track of their information. The WasabiBeans framework is a modern service-oriented architecture implementing the concept of virtual knowledge spaces.

2.2 Multiple Views on a Data Collection

Different applications require different types of data representation. Often each application comes with its own repository containing the related data. In this article the understanding of handling information in cooperative work environments is different. An ideal concept would be to work with one data repository and make these data available in different field of application. Several reasons argue for the use of one data collection instead of creating data redundancies. In order to avoid problems as updating different data collections or the distribution of changes among various repositories, the use of a common data space is essential. When setting up an uniform data collection, it is important to provide different interfaces. Therefore, the concept of virtual knowledge spaces is complemented with free attributes. Free attributes may be used to store application specific values, e.g. the objects' positions for a shared whiteboard application.

Unique and creative concepts, e.g. the pyramid discussion for decision-making processes desire totally different views and representation types (Hampel and Heckmann, 2005). But even the presentation of miscellaneous file formats benefit from different views. Image files might be displayed as a picture gallery, whereas documents are listed alphabetically. There are boundless possibilities to display a data collection or a set of a data collection. It is obvious, that different views are meaningful. Making one data collection available in different applications requires a sophisticated rights management (see section 4).

WasabiBeans, as an implementation of the concept of VKS, intends to be a flexible and adjustable framework. In addition, it provides various interfaces as *Remote Method Invocation (RMI)* or Web services to access the data. In the following these interfaces and the famework's architecture are discussed.

3 ARCHITECTURAL CONCEPT

In this section the framework's architecture is described in general. Afterwards, section 4 describes the authentication and authorization infrastructure and section 5 describes WasabiBeans' extension WasabiPipes to integrate multiple repositories.

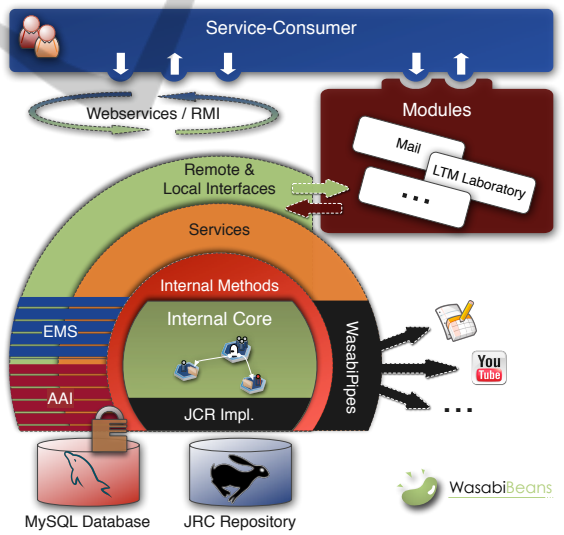

Figure 2: Overview of WasabiBeans' Architecture

Figure 2 gives an overview of WasabiBeans' architecture, which consists of four consecutive layers: internal core, internal methods, services, and the remote and local interfaces.

1. The **internal core** consists of an implementation of the *Java Content Repository (JCR)*[3]. Accordingly, the core provides all functions specified by JCR and is responsible for data storage. Each

[3]http://jcp.org/en/jsr/detail?id=283

Wasabi object is represented by a specific JCR node (compare section 2.1).

2. The **internal methods** are implemented as static methods have two tasks. For one thing they are responsible for basic manipulation of the Wasabi objects, and for another thing for the integration of alternative data sources. More specifically, internal methods work on JCR nodes of the internal core. In addition to the internal methods for retrieval, creation, deletion, and modification of Wasabi objects, there are methods for the transaction management, (section 3.2), the event management (section 3.3) and the version control.
3. The **services** layer consist of stateless session beans. Methods of these stateless session beans supply functions for clients to work with Wasabi objects. For that purpose they use and combine internal methods.
4. The fourth layer **remote & local interfaces** provides interfaces of stateless session beans of the service layer. Thereby, the distinction between local clients[4] and remote clients can be made.

The WasabiBeans framework is an implementation of the concept of virtual knowledge spaces, hence a hierarchical data model is necessary. In consequence of this requirement WasabiBeans uses a JCR implementation rather than the *Java Persistence API (JPA)*[5]. In this implementation Apache Jackrabbit[6] was the first choice, since it is an open source implementation and fully support the JCR standard.

The technology used for managing distributed systems with the WasabiBeans framework server is the *JBoss Application Server (JBoss AS)*[7], which is an open-source technology for building, deploying, and hosting enterprise Java applications and services. Since the JBoss AS provides extended enterprise services including clustering, caching, and persistence, it forms a good base for managing distributed systems. In particular the clustering enables to use multiple WasabiBeans framework for one collaborative system and accordingly to perform load-balancing. JBoss AS is founded on a service-oriented microkernel architecture that ensures all services are accessed, managed, and integrated in a unified and consistent manner. This functionality is essential for the WasabiBeans framework in order to fulfill the design paradigm of service-oriented architectures and thus being adaptable for various field of application.

[4]Local clients are deployed together with the the WasabiBeans framework in one runtime environment.

[5]http://www.jcp.org/en/jsr/detail?id=317

[6]http://jackrabbit.apache.org/

[7]http://www.jboss.org/jbossas

3.1 Main Service Methods

The main service methods provide all functions for clients to operate on the data model. More precisely, these methods allow to work with *Data Transfer Objects (DTOs)*. WasabiBeans has four types of DTOs: object, value, version, and pipeline DTOs. Object DTOs represent Wasabi objects. Value DTOs represent the attributes as well as links. These DTOs are also important for optimistic locking (see section 3.2). Version DTOs stand for a version of a version-able Wasabi object. Pipeline DTOs represent WasabiPipelines (see section 5). Roughly, main service methods can be divided into read and write methods. The structure of write methods is more complex, since they have to consider locks, events, and concurrent writing operations that may lead to failures. The following method invocation of the `create` method of the `AttributeService` shows exemplary how to generate a DTO.

```
WasabiAttributeDTO attributeDTO =
  attributeProxy.create(<"name">, <"value">,
  <dtoOfAffiliation>);
```

A main service method (in this example `create`) can be invoked via a proxy object of the actual service. In the sample above the proxy object is *attributeProxy* of the `AttributeService`. This proxy object can be created by means of a JNDI[8] lookup.

3.2 Transactions

Transactions are absolutely necessary for multi-user systems. Thus, each service method is executed by default within a *Container Managed Transaction (CMT)*. To avoid excessive transaction handling, WasabiBeans can re-use transactions when calling a service method within an already existing transaction. Thereby, existing transactions can be re-used and several service methods may be combined to a single transaction. The following code sample illustrates how a user can combine three method invocations to a single transaction.

```
UserTransaction utx = (UserTransaction)
    jndiContext.lookup("UserTransaction");
utx.begin();
try {
  documentService.create("doc1", roomDTO);
  documentService.create("doc2", roomDTO);
  documentService.create("doc3", roomDTO);
  utx.commit();
} catch (Exception e) {
  utx.rollback();
}
```

[8]Java Naming and Directory Interface (JNDI).

The object to control the transactions has to be requested from the Java EE server via a JNDI lookup. By means of this object the Java EE server can be instructed to begin, commit, or rollback a transaction. All invocations of service methods are performed as part of a single transaction. It should be mentioned that WasabiBeans supports both optimistic and pessimistic locking mechanisms. By default it uses optimistic locking.

3.3 Events

Another key feature of WasabiBeans is the event system. A flexible event system is very important for collaboration and cooperative knowledge management. Users may request a notification if a team member changes a certain file or creates a new document. To create, send, and receive events WasabiBeans uses an *Enterprise Message Service (EMS)*. The current implementation allows the following events:

- *REMOVED:* A Wasabi object was deleted.
- *CREATED:* A Wasabi object was created.
- *MOVED:* An existing Wasabi object was moved within the hierarchical structure.
- *PROPERTY_CHANGED:* An attribute was changed.
- *ROOM_ENTERED:* An user entered a room.
- *ROOM_LEFT:* An user left a room.
- *MEMBER_ADDED:* An user joined a group.
- *MEMBER_REMOVED:* An user left a group.

WasabiBeans uses JBoss HornetQ[9] as EMS respectively as *Message-oriented Middleware (MoM)*. The main challenge was to ensure data privacy for the event system. JMS lack in possibilities to define user specific filters on global JMS queues or topics, yet this is mandatory to prevent users from reading events of others. The obvious solution might be the definition of individual queues for each user. Actually, this would be a secure option, but when specifying event queues for all users (even though they are not logged in), the application server has to manage too many queues. This will definitely lead to performance problems. Figure 3 shows the event system of WasabiBeans. Instead of defining individual queues for each user, there is a *MessageDispatcher* in WasabiBeans, which divides the messages from a global queue to temporal queues (TempDest). To receive change notifications of an object three logical steps are necessary. Firstly, a temporal JMS destination has to be created. Secondly, the new JMS destination has to be registered in the WasabiBeans framework. Thirdly, the user has to be registered as receiver of events for a given Wasabi object. The second and third step are done by means of a stateless session bean.

[9]http://www.jboss.org/hornetq

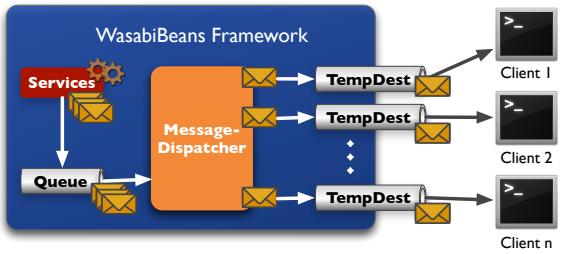

Figure 3: Structure of the Event System in WasabiBeans.

The advantage of this implementation is that the number of JMS queues can be reduced to a minimum without loosing security. Temporal queues are created if a user logs in and subscribes to events of at least one Wasabi object.

4 AAI-INFRASTRUCTURE

Especially for applications in the area of Computer Supported Cooperative Work it is essential to ensure data security. This aspect becomes even more important if the CSCW application is used for safety critical business areas, e.g. knowledge management.

4.1 Characteristics of Access Rights

There exist seven different kinds of permissions in WasabiBeans: VIEW, READ, EXECUTE, COMMENT, INSERT, WRITE, and GRANT. A permission alone does not have any impact, instead it is a property of a complete right. Due to this fact, WasabiBeans uses discretionary access control lists. Every right should have at least the following properties:

- allocation to a specific subject;
- kind of permission;
- allocation to a specific object[10];

This basic structure may be extended. As an additional property, we can consider the following one:

- type of right (allowance or forbiddance);

Here a forbiddance is defined to be more powerful than an allowance. Resulting from this, a forbiddance may overlap an allowance, since more than one relevant right for a specific relation of *Subject–Permission–Object* can be defined. This may occur

[10]In WasabiBeans, the rights are stored directly in the objects they are allocated to.

when setting group rights. In this manner, the property

- status of hierarchy;

is also important. If the authorization function discovers more than one relevant right, the system has to decide which right is preferred.

There is the rule that a forbiddance is preferred to an allowance when belonging to the same hierarchy level. Assuming the levels forbiddance and allowance constitute their own hierarchy, this implicates the effective number of hierarchy levels is twice compared to the number of explicit ones. It should be obvious, that without forbiddance the generation of different hierarchy levels is needless. Resulting a right may have three different conditions: It can be either an allowance, a forbiddance, or not specified.

As an additional property a time interval can be specified for rights in WasabiBeans. Section 4.4 explains the consequences to the authorization infrastructure that are accompanied by temporal access rights. In the following, different criteria that have to be interpreted for the validation of an access right are discusses.

4.2 User Access Rights

An access right always refers to a specific WasabiUser object. It assesses an operation for which the user is authorised on the specific data object or for which he is forbidden. Because of the fact that a WasabiBeans environment may contain plenty of different users, it is not practicable to allocate each conceivable right as a user right. To simplify the right allocation, WasabiBeans offers to specify group rights (see section 4.3).

4.3 Group Access Rights

Within its function, a group right is similar to a user right, with the difference that it is not granted to a specific user, but a whole group of users. Because of this fact, it is sufficient to be a member of the specific group or the corresponding subgroups to gain all associated allowances and forbiddances. Furthermore, there must be a mechanism or rules, that specify which access right is prioritized. This problem may happen in various situations, e.g. if a user is member of two different groups. As a member of group *A* the user may get an allowance for a certain document, but as member of group *B* the user may get a forbiddance for the same document. To solve this problem there is a hierarchy (or rules) that specify which right type has to be prioritized. Figure 4 illustrated the eight different right types in WasabiBeans.

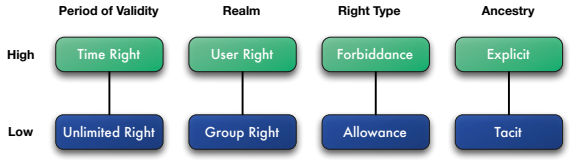

Figure 4: Hierarchy of Access Rights.

By the concept of group rights it is possible to define special fields of responsibility. For example, a group can be created whose members may access on special parts of the object tree to perform well defined operations there. If we want to allocate some users for this field of responsibility, we only need to assign them to this group. A declaration of additional user rights is not necessary.

4.4 Time-based Access Rights

As a difference to the common infinite rights, a time based right is only valid for a certain interval. This means, it is similar to a common right which is expanded by a starting point and an ending point. These two points in time are expressed in the number of milliseconds, which are elapsed since the first January 1970. Referring to this, a time right gets in use if the actual computer time is inside this interval. In a regular case, interferences with other time rights are always avoided by the system. This means, if a new time right is granted, whose interval overlaps at least one of another relevant time right, this interval may be shortened. In an extreme case, it may also be split in two other time rights or totally annulled.

If we abstain from the concept of time rights, a temporary allowance or forbiddance must always be initiated by the administrator to the designated point of time and annulled at another point of time. Thereby, the administrative effort would be elevated and the flexibility of the system would be decreased.

4.4.1 Overlapping Access Rights

As it was already mentioned in this section 4.4, by the initiation of new time rights, there may occur inconsistencies to other time rights which already exist. These inconsistencies are dissolved by preferring the new time right. This means, it may overlap existing rights. To ensure this, before the initiation the system must check every available time right which contains the same user or group name and the same allocated object. Now the interval of the new right becomes compared with the one of the old right. It is $N = (N_1, N_2)$ the interval of the new time right and $O = (O_1, O_2)$ the interval of the old time right. There is the condition $N_1 \leq N_2$ and $O_1 \leq O_2$. Now

we can divide between the following cases (the above cases exclude the below cases, the system works in sequence during the check):

1. $N_2 < O_1$ or $N_1 > O_2 \rightarrow$ There are no difficulties because the two intervals do not interfere.
2. $N_1 \leq O_1$ and $N_2 \geq O_2 \rightarrow$ The old time right becomes replaced by the new one.
3. $N_1 \leq O_1$ and $N_2 < O_2 \rightarrow$ The old time right is shortened on (N_2+1, O_2).
4. $N_1 > O_1$ and $N_2 < O_2 \rightarrow$ The old time right has to be split in two sub-rights with the intervals (O_1, N_1-1) and (N_2+1, O_2).
5. $N_1 > O_1$ and $N_2 \geq O_2 \rightarrow$ The old time right becomes shortened on (O_1, N_1-1).

4.5 Unlimited Access Rights

An unlimited right has got the same function as a time right with the difference, that there is given no interval. Of course there is the possibility to abstain from unlimited rights and to generate each right as a time right. By this case, an unlimited right would be a time right with the highest possible interval. But such a decision leads to a problem: If we generate a new time right, an old, virtual unlimited right which collides with it, has to be split in two sub-rights (see 4.4.1). From this, the following disadvantages result:

- The old, once unlimited right should be deleted $\rightarrow$ Now we have to delete two sub-rights.
- The new time right should be deleted $\rightarrow$ This results in a gap.

As a consequence, we generate two different layers for the differentiation of time rights and clearly unlimited rights. The time based layer should be above the unlimited one, because it can be considered as more special. The figure 5 should illustrate this concept. Another advantage of it results from the fact that we always have the possibility of a clearly abdication from time based rights.

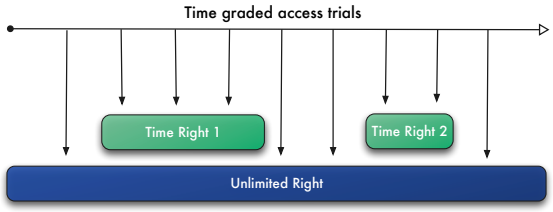

Figure 5: Time-based Access Rights.

4.6 Inherited Access Rights

The adjustment of data inside a tree structure suggests that access rights may be passed from a high into a junior instance, which can be considered as inheritance. If we abstain from this possibility, it is necessary to grant explicit rights for each single data object which increases the administrative effort. In addition to this, on ordinary scenarios we often have got the case, that all objects inside a junior structure have the same or at least very similar rights. The possibility of inheritance should support this.

Basically, we have got the principle that with two or more relevant rights within the same triple relationship (subject, permission, object), always the right which is nearer to the target object has got priority. This means, if there are for example two containers above a WasabiDocument, in which is in each case indicated a right for the same user, the right of the lower container is preferred. If there is even a collision of an inherited right with an explicit right, the inheritance may not occur. For time rights, there again is the principle that the rights with the shorter distance to the specific object may shorten, occlude or divide the intervals of the rights with the larger distance.

The concept of inheritance together with hierarchy for access rights allows to modify rights without high administrative effort.

5 WASABIPIPES: A PIPELINE EDITOR FOR WASABIBEANS

This section describes *WasabiPipes*, a dynamic and adjustable solution to integrate various repositories. WasabiPipes bases an the *Pipes and Filter (PaF)* architectural pattern that divides the tasks of a system into several sequential processing steps. A common definition for this pattern is given by Buschmann et al. *"The Pipes and Filter architectural pattern provides a structure for systems that process a stream of data. Each processing step is encapsulated in a filter component. Data is passed through pipes between adjacent filters. Recombining filters allows you to build families of related systems."* (Buschmann et al., 1996).

Inspired by *Yahoo Pipes*, a service that is normally used to assemble RSS[11] feeds, WasabiPipes is implemented as an extension of the WasabiBeans framework. Nevertheless, this concept to integrate different repositories is not limited to WasabiBeans and

[11]RSS 2.0 Specification: http://www.rssboard.org/rss-specification

can easily be adapted to other purposes and infrastructures. Yahoo Pipes allows to handle (information) sources, mix them up, and modify the data using filters. It does not only support the connection of filter modules but also the connection of single fields of filter modules. Therefore, very complex and flexible pipelines can be assembled. Thus, Yahoo Pipes is a powerful composition tool to aggregate, manipulate, and mashup content from around the web (Yahoo!, 2010). Exactly this understanding of aggregating and connecting information goes along with the intention to integrate different repositories in a complex IT infrastructure to set up a powerful cooperative work and learning environment. Similar to the Web 2.0 mashup concept[12], the intention is to aggregate various repositories to a new high-level collection of information.

The current implementation of WasabiPipes includes the binding of seven different repository types. So far Apache Jackrabbit, Amazon web services™, Google Docs, Flickr, YouTube, common file systems, and databases can be connected using via the PaF architecture. Basically, filters are mapped in WasabiPipes through the abstract class *Filter*, the interfaces *Source*, *Sink*, *ContentStore*, and the classes *Wire*, *Sink.Input*, *Source.Output*.

- **Source:** All filters that deliver data have to implement the Source interface, which procures the method void connect (Source.Output output, Sink.Input input).
- **Sink:** All classes that define inputs; have to implement the Sink interface. The method filter will be invoked by the previous filter and is responsible for the processing logic.
- **ContentStore:** A ContentStore offers a method to retrieve stored data.
- **Sink.Input:** An input of a filter is defined by the Sink.Input class. This class contains a reference to both, the filter and the name of the input.
- **Source.Output:** Analog to the class Sink.Input, the class Source.Output procures an output.
- **Wire:** The class Wire connects an output with an input.
- **Filter:** The abstract class Filter includes basic functions. The method forward can be invoked by any subclass. Afterwards it invokes all dependent filters that are connected with the output specified via the parameter. This class is also checks whether the execution is synchronous or asynchronous.
- **ConcreteSource:** This class has an output and invokes the method fillPipeline, thus the pipeline can be filled data. After completion this method invokes the forward method of the superclass to forward the data within the pipeline.
- **ConcreteFilter:** This class defines two inputs and two outputs. Within the filter method data are forwarded from input1 to output2 and from input2 to output1. The inputs and outputs are crossover connected.
- **ConcreteSink:** Finally, this class dumps the data of both inputs. Additionally, it reports which input passed the data.

For the implementation of WasabiPipes the JavaScript library WireIt[13] was used. WireIt allows to store the wiring states in the *JavaScript Object Notation (JSON)*[14] format. A sample JSON description of a MIME type filter is given in section 5.3.

5.1 Concrete Filter Implementations

To create a better understanding of filter implementations, the *MimeFilter* and the *YouTubeSink* are discussed exemplarily below.

The *MimeFilter* consists of one input, two outputs (MATCH and NOMATCH), and a field to specify the MIME type. The method *filter* compares the MIME type of the document and the one defined as filter parameter. In case of a matching MIME type the data are forwarded to the MATCH output, otherwise to the NOMATCH output. Afterwards, the method *forward* of the super class *Filter* is invoked to notify all dependent filters about the result.

The *YouTubeSink* allows uploading videos directly from WasabiBeans to YouTube. However, the upload isn't straightforward and needs some special parameters. For authentication purpose YouTube requires a *developerKey*, a *consumerKey*, and a corresponding *consumerSecret* (a private key). The keys can be requested on the Google website. In addition an *accessToken* with a matching *accessTokenSecret* for the user is mandatory. YouTube provides an OAuth service to afford this token. Unfortunately, YouTube offers read access only; hence the content is not directly accessible through WasabiBeans. Accordingly, the *getContent* method is not implemented yet. Other repositories, such as the Amazon web services™, don't lack of a method to retrieve the content. In that

[12] A Web 2.0 mashup is a (re-)combination of existing services to an aggregated new service that provides additional functions by combining the functionalities of the original services.

[13] http://javascript.neyric.com/wireit

[14] JSON is a lightweight data-interchange format. Online available: http://www.json.org/

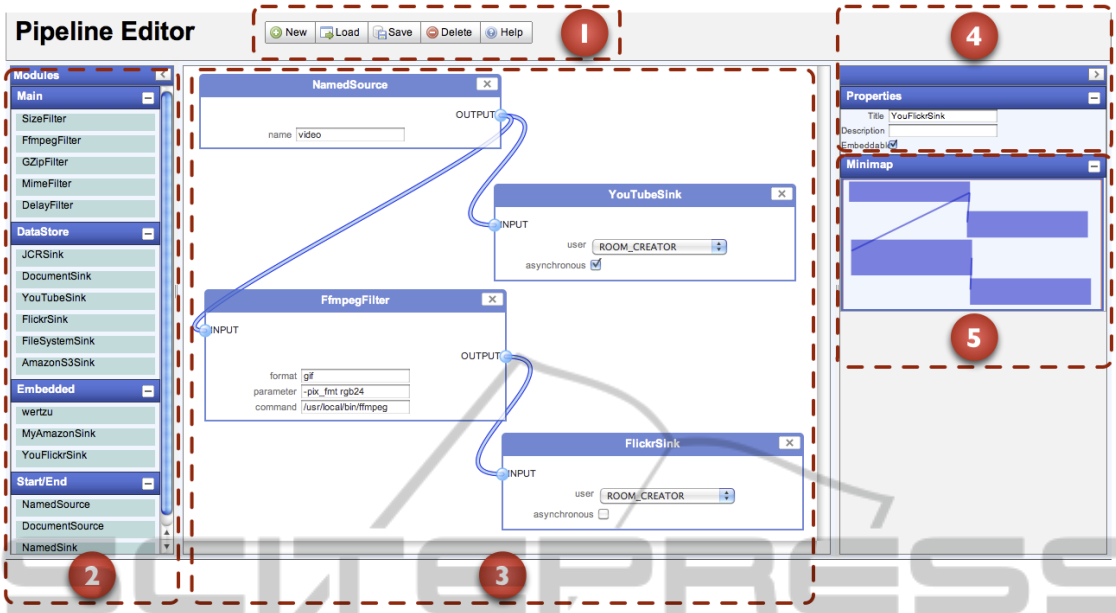

Figure 6: The Pipeline-Editor.

case we can request the data directly from the integrated repository and show it within WasabiBeans' GUI.

5.2 A Graphical User Interface to Connect Repositories

WasabiPipes is accessible via the WasabiBeans administration GUI. Figure 5.2 shows the editor, that allows creating and managing pipes for specifying the data flow. The WasabiPipes editor consists of five main parts.

1. A toolbar to create, load, save, and delete pipes. Additionally the interface supports a help button for unpracticed users.
2. In the left area the modules are available. Modules are filters, data storages, embedded, and start/end components. Users can reuse these modules via simple drag-and-drop action. It should be mentioned, that it is possible to save pre-defined pipes as embedded modules and use them in more complex configurations. Thereby, different pipes can be nested and a clear overview of the entire configuration can be guaranteed.
3. This is the main working area to arrange the components and specify the sources, filters, and sinks.
4. In this area users can add some additional meta data, e.g. title and description to the designed pipe.
5. To ensure a clear overview of large pipes we also offer a mini map, marked with 5.

5.3 JSON Description of MimeFilter

```
{
  "name": "MimeFilter",
  "category": null,
  "container": {
    "xtype":"WireIt.InOutFormContainer",
    "inputs":[{"name":"INPUT"}],
    "outputs":[{"name":"MATCH"},
      {"name":"NOMATCH"}],
    "field":[{"type":"string",
      "inputParams":{
        "label":"mimeType",
        "name":"mimeType",
        "required":true,
        "value":null
      }},{
      "type":"boolean",
      "inputParams":{
        "label":"asynchronous",
        "name":"asynchronous",
        "required":false,
        "value":false
      }}
    ]
  }
}
```

6 CONCLUSIONS

Since collaborative activities are very similar in various fields of application, a CSCW system should be flexible and adaptable for the usage in different fields of application. Although there are common CSCW tools (e.g. Blackboard[15] and Moodle[16]) there is a demand for integrating existing applications to achieve system convergences (Papazoglou and van den Heuvel, 2007). WasabiBeans is a flexible framework to maintain the implementation of cooperation support environments by integrating existing applicaitons. In addition, WasabiBeans provides functions to work efficient in virtual knowledge spaces. Due to the flexible SOA, WasabiBeans is ready for the next generation of CSCW applications. The object model with free annotations is optimized for adjustable knowledge management.

Web 2.0 evolutions have placed a new focus on current CSCW developments. New technologies (e.g. RSS feeds, tagging, etc.) have to be integrated in an existing architecture to serve the demands of semantic knowledge structuring. The presented WasabiBeans framework is able to adapt multiple Web 2.0 services and systems in flexible and scalable manner. Beyond, the extension WasabiPipes, a pipes and filter architecture, allows the integration of heterogeneous repositories. By means of encapsulated functionality and isolated filters it was possible to provide highly parallel operations and data transfer. In particular, when integrating external services this is beneficial. Thereby, the execution of further filters is enabled, even if the system is waiting for the completion of an upload. During the Web 2.0 era more and more services were offered and APIs are ubiquitous in the Internet. The most prominent sample is Google with its services (YouTube, Google Docs, ...), which can interact by way of using the common interface gdata. Cooperation and knowledge management supporting systems have to meet the challenge of integrating this wide range of different repositories, since knowledge management bases on information sharing and a common persistent data repository.

REFERENCES

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (1996). *Pattern-oriented Software Architecture - A System of Patterns*. John Wiley & Sons, Chichester.

Dengel, A. (1994). *Künstliche Intelligenz - Allgemeine Prinzipien und Modelle*. BI-Taschenbuchverlag, Mannheim.

Eßmann, B., Hampel, T., Goetz, F., and Elsner, A. (2006). Embedding collaborative visualizations into virtual knowledge spaces. In *7th International Conference on the Design of Cooperative Systems*, page 3340, France, Provence.

Greif, I. (1988). *Computer-Supported Cooperative Work: A Book of Readings*. Morgan Kaufmann Publishers, Inc., Santa Mateo, CA, USA.

Hampel, T. (2001). *Virtuelle Wissensräume - Ein Ansatz für die kooperative Wissensorganisation*. Dissertation, Universität Paderborn.

Hampel, T. and Heckmann, P. (2005). Deliberative handling of knowledge diversity the pyramid discussion and position-comentary-response methods as specific views of collaborative virtual knowledge spaces. In *Proceedings of Society for Information Technology and Teacher Education, 16th International Conference Annual, SITE 2005*, Arizona, USA.

Hampel, T. and Keil-Slawik, R. (2001). sTeam - designing an integrative infrastructure for web-based computer-supported cooperative learning. In *Proceedings of the 10th International Conference on World Wide Web*, pages 76–85.

Hibbard, J. (1997). Knowing What We Know. *InformationWeek - The Business Value of Technology*.

Licht, T., Schmidt, L., and Luczak, H. (2003). Goal awareness in distributed cooperative work settings. In Luczak, H. and Zink, K. J., editors, *Human Factors in Organizational Design and Management VII (Aachen 2003)*, pages 329–334, Santa Monica. IEA Press.

Liebowitz, J., editor (1999). *Knowledge Management Handbook*. Boca Raton, Adelphi, USA.

Papazoglou, M. P. and van den Heuvel, W.-J. (2007). Service oriented architectures: Approaches, technologies and research issues. *Very Large Data Bases*, 16(3):389–415.

Prinz, W. and Gross, T. (2001). Ubiquitous awareness of cooperative activities in a theatre of work. In Bode, A. and Karl, W., editors, *Fachtagung Arbeitsplatzcomputer: Pervasive Ubiquitous Computing*, pages 135–144. APC.

Yahoo! (2010). Yahoo! pipes – http://pipes.yahoo.com/pipes.

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

[16] Moodle is a Learning Management System (LMS), http://moodle.org/.