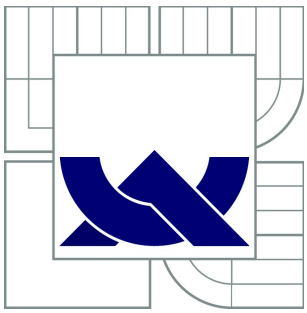

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

COLLABORATIVE TEXT EDITING IN A PORTAL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JÁN KORČÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK KOČÍ, Ph.D.

BRNO 2012

Abstrakt

V tomto texte sa zameriame na populárnu koncepciu kolaboratívnej tvorby dokumentov. Predstavíme si myšlienku využitia tohto mechanizmu v rôznych oblastiach rozhodovania, popíšeme si koncept a princíp fungovania. Následne si predstavíme a rozoberieme portály a portletovú technológiu, ich výhody a využitie. Cieľom práce je implementácia kolaboratívneho editora s využitím knižnice pre prácu so zmenami v dokumentoch s perzistentnou a aplikačnou logikou na platforme JEE a vytvorenie jednoduchého portletu pre túto službu.

Abstract

In this paper we will concern on popular concept of collaborative editing. We will introduce the idea of leveraging this mechanism in a diverse areas of decision making, we will denote the concept and principle of work. Then we will introduce and discuss portals and portlet technology, its advantages and use. The objective of the work is an implementation of collaborative editor leveraging the library for management of changes on documents with the persistence and application logic based on JEE platform and creation of simple portlet for this service.

Kľúčové slová

kolaboratívna tvorba dokumentov, portal, portlety, Etherpad, Etherpad lite, Java Portlets, JPA

Keywords

collaborative editing, portal, portlets, Etherpad, Etherpad lite, Java Portlets, JPA

Citace

Ján Korčák: Collaborative text editing in a portal, diplomová práce, Brno, FIT VUT v Brně, 2012

Collaborative text editing in a portal

Declaration

I hereby declare that this thesis is my own work that has been created under the supervision of Ing. Radek Kočí, Ph.D. Where other sources of informations have been used, they have been duly acknowledged.

.....

Ján Korčák

23.5.2012

Acknowledgements

I would like to thank Ing. Radek Kočí for the time he spent during discussions with me. He offered me valuable advices, consultations, which has helped me a lot. Moreover, I would like to thank Mgr. Michal Vančo from the Red Hat company for his encouragement and cooperation on project. Finally, I would like to thank my family for their support during my work on thesis.

Contents

1 Introduction

There is a well-known saying that two heads are better than one - this states that when more people are solving the same problem together, they can be more successful in finding the solution than when the same problem is being solved by fewer people. This idea can be used in spheres where the focus is put on a quick decision or where you have to bring out the solution in a short time. When working on a project within a group of people the complex task is usually decomposed into a number of smaller tasks. Each of these tasks can be processed by a single person. If the task can be processed in a parallel, it can speed the processing of whole task; otherwise the process could be slower. However, we will not talk about a decomposition in this work. The collaborative editing is focusing on the tasks, when we can provide more quickly and even a better solution when more people are cooperating on the task in the same time. There can be founded some examples of practical use of this idea even in the same platform we will talk about later – text editing.

In IT the idea of collaboration on the editing of single document is similar to the concept of well-known programming method called extreme programming, where mostly a pair of programmers (therefore also called pair programming) are creating one file together. This kind of work is performed in a way of currently editing the file by only one programmer, while the second programmer is watching the process, supporting writer by thinking of the code and control the writer. During the process the roles can change, so the supervisor programmer may take the role of execution programmer and former execution programmer may take a watching role. This concept makes programming more agile as it lets the both of programmers concern on the part of the process more intensively; this could make the development process more effective.

The objective of the thesis is to examine the field of collaborative editors, especially the product called Etherpad Lite and provide an implementation of such application with the JEE and portlet technology.

Chapters 2 and 3 will provide an introduction to the technologies we will work with. First there will be described concept of collaborative editing, analogy with known technologies and implementation of Etherpad collaborative editor, resp. Etherpad Lite. Then in third chapter we will introduce portlet technology and portals. There will be discussed fundamentals of portlets introduced in Portlet specification.

Chapter 4 will provide an inspection to the Etherpad Lite implementation of collaborative editor; its domain, main components providing core features and processes performing within the application.

The analysis and design process of application will be presented in the chapters 5 and 6, where

the 6th chapter will discuss the persistence tier and 7th application tier.

Chapter 8 is dedicated to the presentation of analyzed optimizations and extensions for the application, that could be implemented in the later development.

The last chapter is conclusion of the project.

After the bibliography and figure registry the paper has a 3 appendixes with how to start the application and descriptions of the database model used.

2 Collaborative editing

In this chapter we will look on the concept of *collaborative editing* (CE) and introduce actual implementations. As the site of one from CE implementations, the SubEthaEdit project, says[1], the idea of CE is not so recent since it has been researched during the years. There can be found a number of applications offering such a functionality that differs in the pack of extensions each of them provides on behalf the core CE function. However, we will be discussing an implementation of CE, we should define the term of the *collaborative editor* first.

2.1 Introduction to the concept

2.1.1 Definition of CE

Although, there is not an unified definition of the concept, we need to define the CE for our purposes, so that we could build upon it. We can start with the definition provided by the Wikipedia Foundation:

A collaborative editor is a form of collaborative software application that allows several people to edit a computer file using different computers.

It describes the application that enables the editing of the specific file by a group of people each of them sitting in front of different computer. However, this definition does not specify the time dimension and thus it can cover wide range of uses - you can surely imagine working with a shared file using tools such as secure-shell, FTP or even by exchanging the updated file. This is however only about sharing and there cannot be any talk about an effectiveness; the whole idea is almost a rubbish, not to mention the complication of distribution within a large group of users.

What the CEs brings is not only the ability to share a document, but also to edit the document concurrently, during the other people works on it at the same time. Such a behavior must be supported by the sufficient synchronization mechanisms. We will dive deeper into this problem in the section concerning the similarities of CE with the revision (version) control systems.

2.1.2 Division of CEs

To push the description of CE more far, we have to provide a division. The most important attribute with the very significant matter, as it affects the users, is a way of synchronization; this splits the CEs into two main subgroups:

Real-time collaborative editor (RTCE). Synchronization is invoked on every change of text. Users can edit the same file simultaneously and they are provided with the latest version of the document in their editors in almost every second of work as the editor content is immediately updated when remote

1 More informations can be found on this site: http://www.codingmonkeys.de/subethaedit/

change occurs.

Non real-time collaborative editor. Synchronization is triggered with during the saving or on different action occures, depending on the strategy of concrete editor. Users therefore can edit the same file simultaneously, however, they are not exposed with the fresh content in such small intervals as it is in the RTCEs. Its behavior is more like that of *revision control systems*.

We will discuss the first type of CE, but before providing a deeper description, we will look at probably more familiar technology to make some intuitive background.

2.2 Implementations of CE

2.2.1 Google Docs

One of the most popular collaborative editing programs is probably Google Docs word processor called *Writely*. It was founded by the Silicon Valley startup called *Upstartle* and during its existence has attracted thousands of users. The main breakthrough, however, came with the Google's acquisition of Upstartle. Big advantage of the Google here was that the editor was included into the portfolio of Google services. In a short time from adopting, Writely has made a millions of the Google users.

However, what Writely provides is non real-time collaborative editing - you may cooperate on writing of the document, but you can not see the state of the document in a real time. The synchronization is triggered on every save action of the document, so you can not see the real progress in it. Another thing lacking in comparison to the RTCEs is that the editor does not provide user distinguishing by color; therefore you can not identify the authorship of the exact slice of text within the document.

2.2.2 Real-time collaborative editors

Presentation properties

There are some characteristics that are common for many of these products and by which we could identify them. Some of them are:

Real-time concurrent editing. The text can be edited by more than one writer in a time which is the main idea of this concept; the text typed by another co-writer is shown immediately character by character. The process is non-blocking and all conflicts among the authors currently editing the text are solved automatically. This behavior is assured by the conflict solving mechanism, which will be described in the section 2.4.

User distinguishing. Since we can edit the same file with the other person or even a group, we may want to distinguish the users working on document, so it will be possible to watch the text we are

editing and identify text already written by us - otherwise it could be difficult to watch the changes in text. RTCEs therefore provides each user with a color that distinguishes the text written by himself from the other.

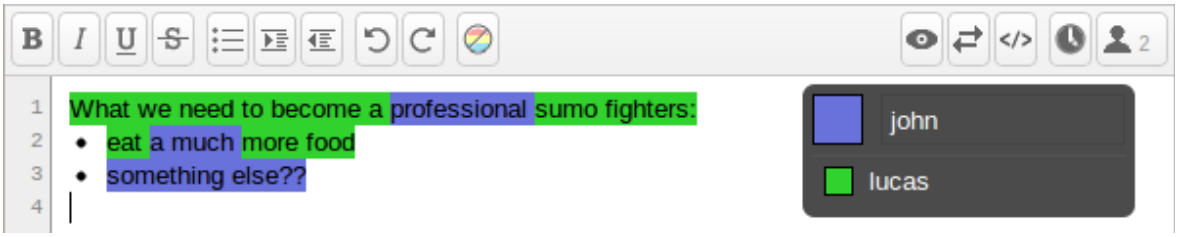

Figure 1: User text distinguishing in the Etherpad Lite CE

Unbounded history. The CEs works as a revision control systems in a way it deals with changes. Every change is encapsulated as an increment to previous state. This encapsulation is called a changeset. Each changeset is saved and since it exists also an inverse algorithm, it is possible to walk through the created changesets and provide the history revocation. Storing the every change then enables an access to even the oldest state of the document.

2.3 The Etherpad

This work concerns on the concrete RTCE product with motivation of leveraging this technology to provide an integration within the another platform. We could therefore put down some start informations about the technologies we will work with.

2.3.1 Old Etherpad

The Etherpad is a project of AppJet Inc., which was later acquired by Google. It is a web-based application providing collaborative editing of documents hosted on network. The main functions it provides are:

- real time collaborative editing of documents,
- distinguished users texts,
- creating and cloning pads,
- import and export to and from a different types of documents,
- inserting and embedding the files into pads,
- uploading images into pads
- and more.

Distribution and requirements

The Etherpad is an open-source product that can be downloaded and hosted by the user needs. It is possible to run it on both the external and local server and provided for use in the company or even within a group of people without any IT infrastructure built. What is needed, are the Internet connection and the Etherpad installation with a pack of open-source free downloaded packages.

Since the Etherpad was made in collaboration of technologies: Java, Scala, JavaScript and macro processor M4, required runtime environments and libraries must be installed on the target machine.

Application needs to run on the Open JDK 6 distribution of Java Virtual Machine (JVM), therefore this runtime environment must be installed. The application is configured on every startup with the build script to set the proper paths, so that the JAVA_HOME is targeted to the right environment.

The server side of application uses the MySQL database to store a content, so the MySQL database server must be installed too and the MySQL connector must be presented in order to enable the Java application to connect to database.

EasySync protocol

In a collaborative editing you have a number of editor instances - each for user working on another computer. The critical task that must be provided by server is maintaining the consistency among the different document replicas, so the users have the latest version of the document in their editors. This process must be non-blocking, because the users don not want to wait for the end of synchronization. That is why the Etherpad is using the protocol named *EasySync*, which describes identification and extraction of the changed part of the document and its encapsulation within a changeset object. EasySync also provides functions for applying a changeset to the document and a strategy for conflict solving when changesets from different users meet in the same time.

2.3.2 Etherpad Lite

Disadvantages of Etherpad are that it requires relatively high software and hardware resources. The installation is not very user-friendly, too. Therefore there was a motivation to make a lighter implementation of Etherpad that could be more accessible than the original one.

The project named Etherpad Lite got started afterwards. It was successful and has brought even more invention since it was built only with use of single language - JavaScript and using a number of JavaScript frameworks helping to build a server-side of the application. The Etherpad foundation adopted the Etherpad Lite and made it an ancestor of the origin Etherpad since it was practically its upgrade. Nevertheless, Eteherpad Lite does not cover the whole functionality of old Etherpad.

Currently you can see a reference to Etherpad Lite project on the official Etherpad web page with original inventors encouragement to use it instead of the old one. The in-depth analysis of this implementation will be presented later in the paper.

2.4 Collaborative editing systems vs. revision control systems

We have mentioned earlier that a bit of similarity between the *collaborative editing systems* (CES) and *revision control systems* (RCS) used in a software development can be found.

RCS, also known as version control systems are systems which manage changes to documents, computer programs, or any other editable files. From the view of the software development, RCS clarifies the development of projects involving more developers by providing tools for maintaining consistent state of projects and revision control.

For the sake of understanding the concepts used within the CES and collaborative tasks in general, it is good to look at the RCS familiar processes and describe them briefly.

2.4.1 Changesets and revisions

It has been already outlined what the *changeset* is, but we should try to provide more illustrative description. Changeset definition can diversify among the different RCS and the specific implementations. The main principle is, nevertheless, quite simple. Basically, the changeset is a set of changes made on document by concrete user. These changes are packed in one unique named object that can be applied to a document as a patch. This patch can be then applied to the document in order to update the state of document and thus provide a new document. The extraction of these changes is held within the object Changeset. If we look on the whole document from the creation along to the current state, it can be defined as application of ordered collection of all changesets created on the document to an empty document. After the application of changeset, the current state of document is called *revision* and is identified by a unique *revision number*. This is illustrated on the next figure.

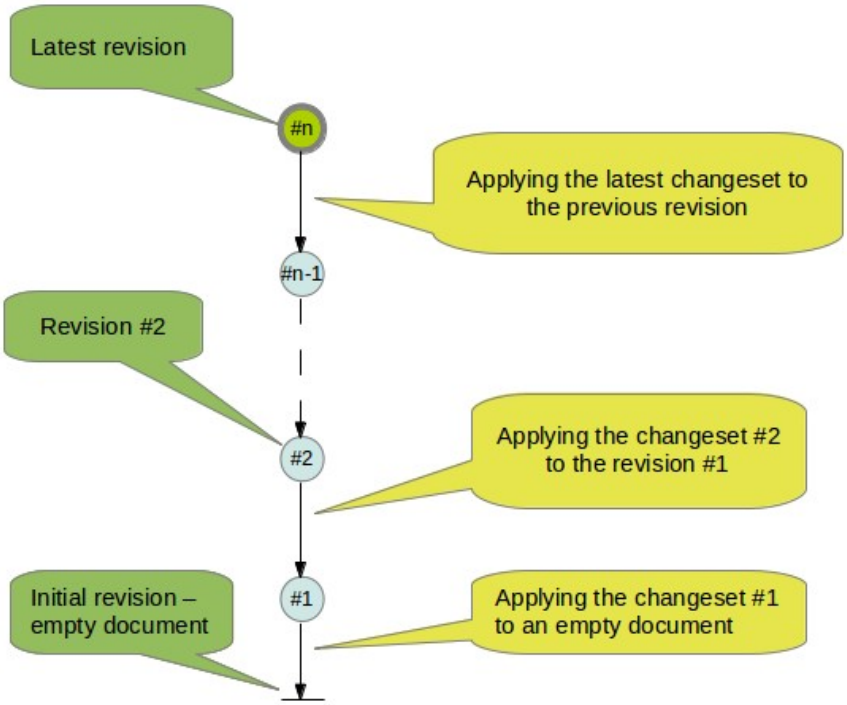

Figure 2:Building a document by application of changesets

2.4.2 Synchronization

In the CES the most crucial part of the whole system is the changeset management. It provides synchronization for the users so they can have the latest state of the document in their editors while writing. That is why the situation where two users spent an hour writing the same text on the same document cannot happen. This is a synchronization issue which is one of three things they have in common with the revision control systems.

However, the synchronization is not working in the same way. In the RCS you have to put a request every time you want to synchronize your working copy with a server state. ON the other side, in the CES this must be done when the users work; it must happen automatically and most frequently to ensure that the writer has the most actual version and therefore he will not produce a different document as server copy. In a next figure a concept of synchronization in CE is shown.

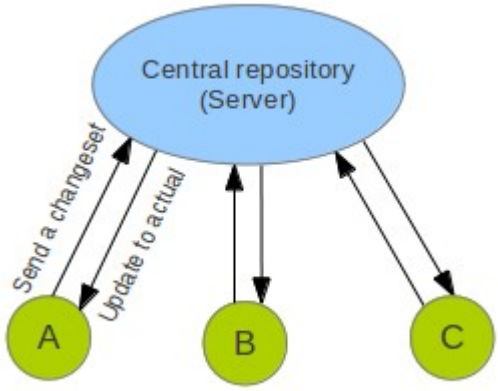

Figure 3: Users A, B and C are synchronized with a server copy.

2.4.3 Conflict solving

Second functionality that can be seen in both RCS and CES is conflict solving. This is very important part of the CES as the whole mechanism must be automatic and the conflicts would occur more frequently as in a RCS, therefore the must of solving every change manually is almost impossible. There is even no objective need of it. Resolving this issues automatically provides that the users could not be blocked from editing and requested to conflict solving because of bad synchronization; concrete mechanisms will be discussed later in this paper. Now we will provide an example situation when you need a conflict solving and describe the asset of such mechanism.

Imagine a situation when there are two users editing the same document. If we abstract the behavior of such editor and say that the synchronization of users editors contents is provided in bigger intervals or manually at the saving of the document by user and there is no conflict solving mechanism.

Each of users now have made a different modification on the same file. The users now have different documents in their editors without knowing of the other ones modifications. The problem raise when both tries to update the current server copy. As from the algorithmic view there is not possible that the two updates came in the exactly same time, the server first applies one of them and tries to update the document on the server. The operation could finish successfully, because the changeset was made as an increment of the previous state of document which is equal to that stored in a server. But once it tries to apply the second changeset, he meets the conflict. However, this changeset was made as increment to the previous state of document, and therefore it cannot be applied because the update could bring the document not in consistent state. The situation is illustrated on the next figure.

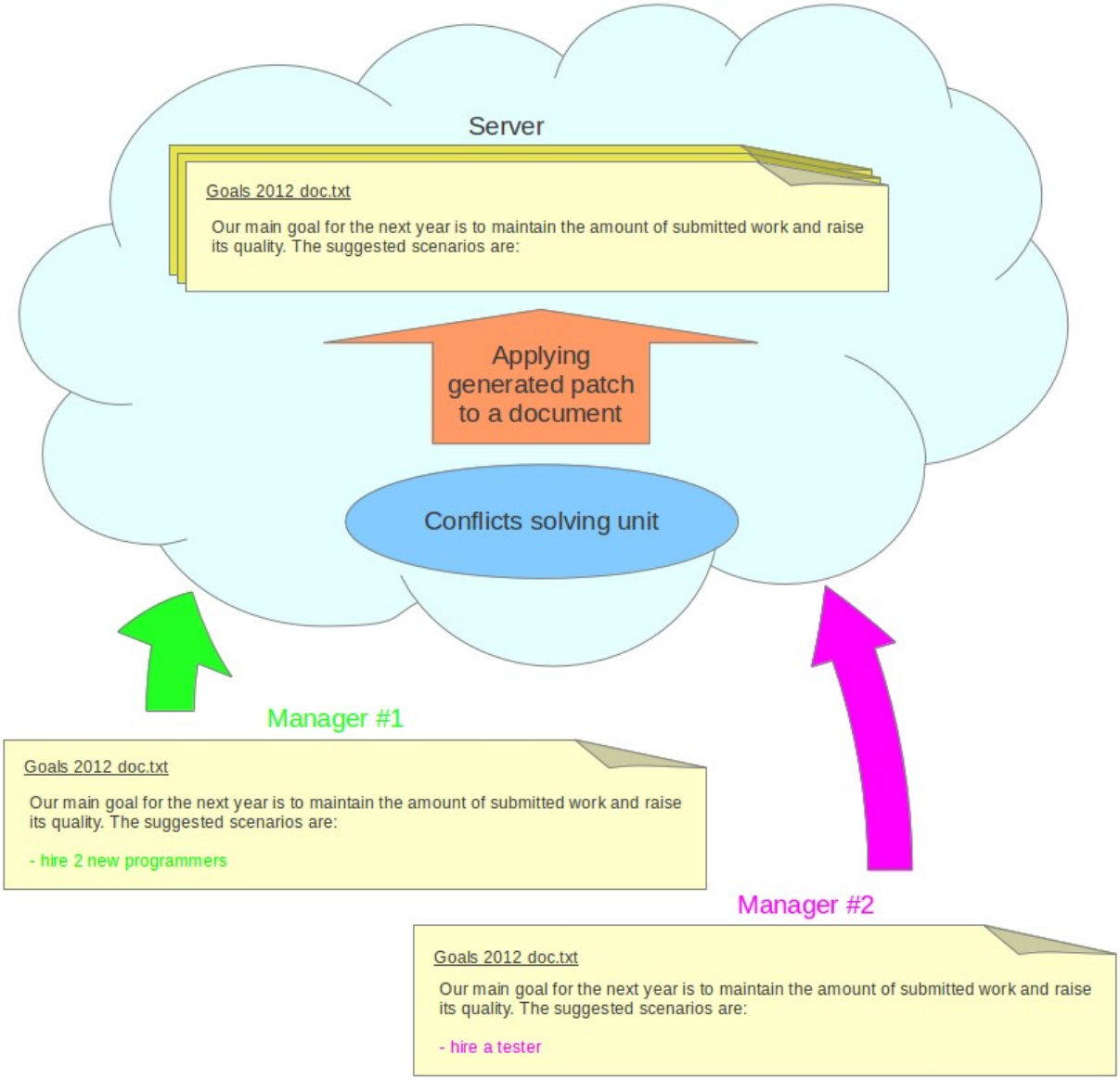

Figure 4: Example: Production of conflict: 2 managers are writing down the possible scenarios to meet the specified goal.

However, the conflict solving is present and the specified situations can be solved by merging the actual state of file with a given changeset. Nevertheless, there may occur situations, when the conflict solution can not be find automatically by a system. In such situations the changeset can be refused and user is requested to backup its state, update its copy to the current document and until then apply the change. But this scenario could be find just in revision systems, so we can suppose that after the generation of merge its application file in the example could look as follows:

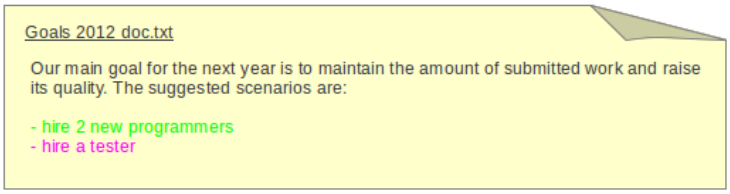

Figure 5: Example of conflict solution for a situation denoted on the picture in previous figure.

History

Finally, the version systems and CES also provide a history of changes so the user can every time, when he needs, browse through the revisions and inspect the changes or even dynamically revert to the specified older version. The file is then put into state of this revision without need to clicking back to the wanted state.

3 Portals and portlet technology

3.1 Portal

If we want to discuss portlets and portlet technology, we will not omit a mention about the term portal which is a main playground for portlets and on which whole technology is built.

3.1.1 Common definition of a Portal

A *portal* or a *web portal*, since we will discuss portals used especially in the ground of web sites and web applications, is a well-known concept used in the area of graphic user interfaces. There were no formal definition of it until recently, however, there are some that can describe it sufficiently for us.

On the Wikipedia.org we can find really simple definition of what a web portal is:

A web portal is a web site that brings together information from diverse sources in a unified way.

It can be also described as a collection comprising a number of small web applications each of them has its own scope and context; the idea is to collect and enable co-existence of multiple applications

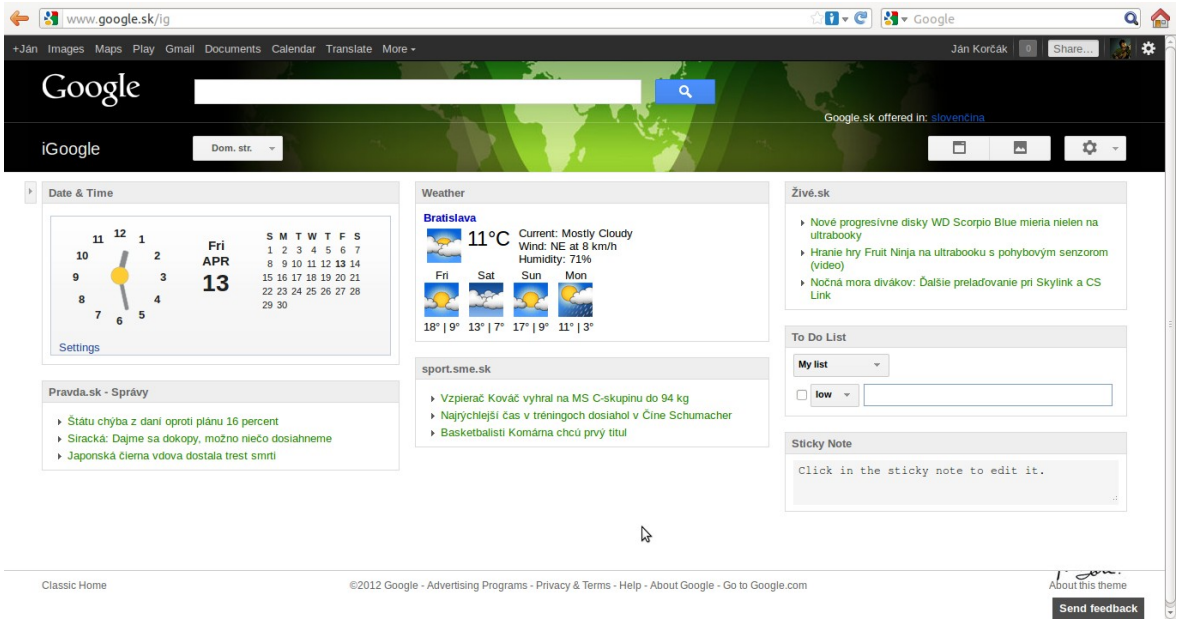

Figure 6: Example of a web portal: iGoogle by Google.

within the one web (portal) page. This is rather comfortable solution as compared to standard approach, where you have to use many different sites to gather all the needed informations. On the next figure you can see Google's portal *iGoogle*, which can be referenced as a common portal, as we have

just introduced it.

3.1.2 Definition of a Portal by the Java Portlet API's

The definition provided by Java Portlet Specification[2] is little bit different from that commonly used one. Within the Java portal the mechanisms and principles are more rigorous and therefore we can surely write down the main characteristics and abilities of Java portal. According to Java Portlet API, the portal supports features like:

Content aggregation. Each portal page can concentrate the modules or services with a different functionality – providing different output.

Authentication. User have to submit his credentials to confirm his identity on the portal in order to get access to the portal content, the advantage of portal is, that it provides single sign-on capability, so you have to authenticate only once, to have access to all the portlets within the portal page.

Personalization. Portal identifies user within a session or within a whole application scope and provide him services for him and by his custom preferences.

Customization. Provide user with ability to have different preferences for the view and functionality.

3.2 Portlets

We have introduced a portal as a content aggregating concept of web application that concentrates on one web page more applications. These applications are called *portlets*. In the next topic we will start with a formal definition of what the portlet is.

3.2.1 Definition of a Portlet

The definition provided by Sarin in Portlets in Actrion says:

A portlet is a pluggable user interface component that provides a specific content, which could be a service or information from existing information systems. Portlets provide the user interface of the portal by accessing distinct applications, systems, or data sources and generating markup fragments to present their content to portal users.

There we have a portal in role of a container or a platform for the small applications – portlets. Portlets act as windowed web applications within the portal and each window in a portal web page represents a portlet. They can be personalized and customized, they supports aggregation of content and also authentication.[1]

The figure 7. illustrates how can the portal page look like and how the portlets are represented

2 There are 2 Portlet specfications: JSR-168 and JSR-286 (referenced in document as [9] and [10]).

within the portal page.

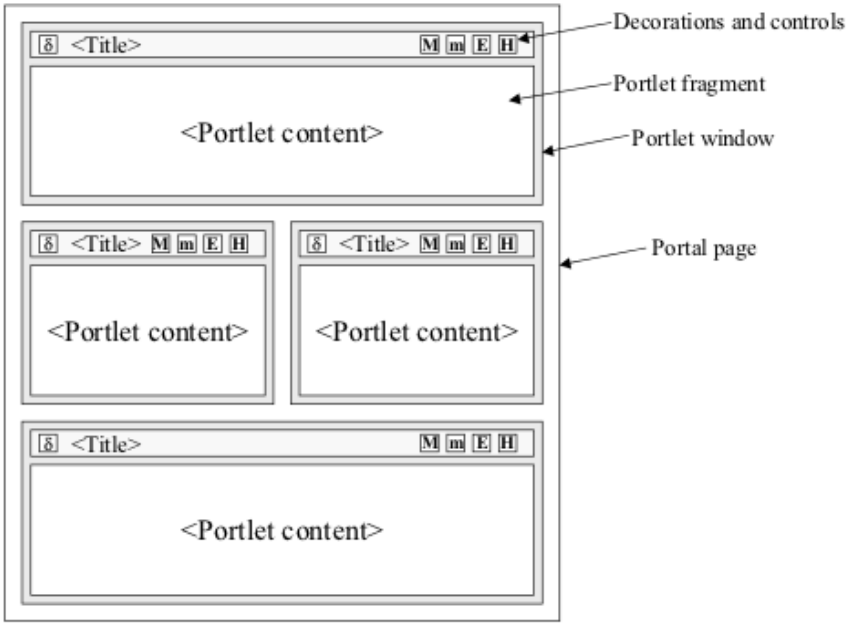

Figure 7: Portal page decomposition. [10]

3.2.2 Java Portlet API

The two specification for Java Portlet API: JSR-168 for Portlet version 1.0 and updated specification JSR-286 for Portlet version 2.0 were introduced.

The definitions of both portal and portlets was made in the JSR-168 and later upgraded in the JSR-286, introducing new features for Portlet 2.0. The 2.0 specification addresses most of the frequently required features of portlets that were missing from the 1.0 specification, such as resource serving, inter-portlet communication, and portlet filters. However, most portlet containers support both specifications, and the 2.0 specification is backward compatible with the 1.0 specification; and therefore we can run Portlet 1.0 applications also within the most new Portlet containers.

3.2.3 Aggregation and SOA

As the portal provides a container for locating and managing diverse portlets, it enables to concentrate whole bunch of services or applications into one page. This is one of the main motivations why to develop with portlets. The second significant advantage of portlets is their ability to interact with other portlets in the portal. This is provided by the process called inter-portlet communication.

The capability of communication between the portlets in the portal can be very likely used in *Service-oriented architecture* (SOA), as it makes it possible to build new applications leveraging the

existing services. In portlets can be seen even an attitude when the applications provides new services only by composing existing ones; these are known as *mashups.*

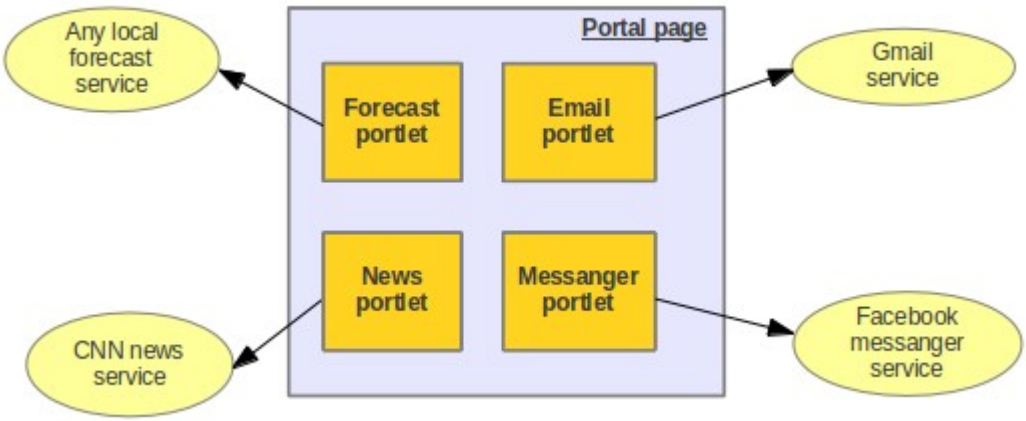

Figure 8: Illustration of content and service aggregation within the portal page.

3.3 Portlets vs. Servlets

In a number of books concerning on the portlet technology, there is often portlets nature described through the comparison with the fundamental component of all Java web applications, *servlet*. It is not a coincidence, as the portlets are in fact extension of the servlet specification and so we will provide this comparison too. First we will introduce a common attributes for both components and then we will chronologically pass to the portlet specific ones.

3.3.1 Portlets relationship with servlets

What have portlets and servlets significantly in common is, that they are both a web components. And as the web components their life cycles are managed by its containers. The portlets containers responsibilities are quite the same as the servlet ones, however, the Portlet API is an extension to the servlet specification, which also means that a portlet container is also, by definition, a web container. Portlet container's function is loading and instantiating a portlet, initializing portlet instance, directing requests to it and finally destroying the portlet instance. The second fundamental attribute is that both of them generates dynamic web content via the request-response paradigm.[2][9][10]

3.3.2 Differences between portlets and servlets

According to publication from Mr. Richardson and col., there can be found following differences between portlets and servlets:

Portlets generates only fragments. On oppose to servlets that generates whole web documents, portlets generates only a markup fragments for a web page.

Not bound directly to the URL. Unlike servlets, portlets are not bound directly to the URL.

Sophisticated request scheme. Portlets have more precise request scheme consisting of 4 types of requests: *action, render, event* and *resource*.

Standardized set of modes and window states. For portlets there are predefined different modes, that defines their operating context and rendering rules and window states, that defines the portion of space within the portal page area portlets will take.

Among the differences the portlets differs from servlets by providing some features, that servlets does not:

Management of configuration informations. Portlets can easily access and store the configuration data thanks to its sophisticated management.

Access to user profile information. Portlets went beyond the basic user and role information providing in the servlet specification.

URL rewriting. This means an ability of creating hyperlinks within the portlet context, which are independent from the portal server implementation.

Two session scopes. Portlets can store a transient data within the two different session scopes: *application-wide scope* and *portlet private scope.* These *w*ill be described later in the section concerning on Portlet Session.

Inter-portlet communication. Portlet can communicate with the other portlets by the sending and receiving events.

Multiple portlet instances. The same portlet can exists within the same portal page multiple times.

And there are also functionalities that portlets on oppose to servlets does not provide. The functions that portlets lacks are:

Encoding. Portlet cannot change the charset encoding of the render response.

Access to URL. Portlets does not have access to the URL with which the client initiated the request on the portal.

3.4 Portlet infrastructure

This section will describe some portlet basics such as an description of main components of portlet technology, their responsibilities in managing the life cycle of portlets and finally, portlet options, that could determine its view or behavior in a portal page.

3.4.1 Components

Portal page

A portal consists of web pages, in portal application called *portal pages.* These web pages includes some header, footer and the main content consists of variable number of portlet applications.

Portal server

A *portal server* handles the contents provided by the portlet container and creates the portal page as a sequence of HTML code. Portlets are located within the portal page by the portal server that applies the specified layout to them.

Portlet container

The main component within the portal page is a *portlet container*. In it a portlets are deployed. This is similar to servlet container, although, a portlet container is responsible for managing the communication between portlets and sending generated fragments to the portal server. It invokes life cycle methods on the portlet instances and provides a runtime environment for them.

The portlet technology builds upon the servlet technology, since it's portlet container is practically an extension to the base servlet container also known as *web container* (drawn in the figure

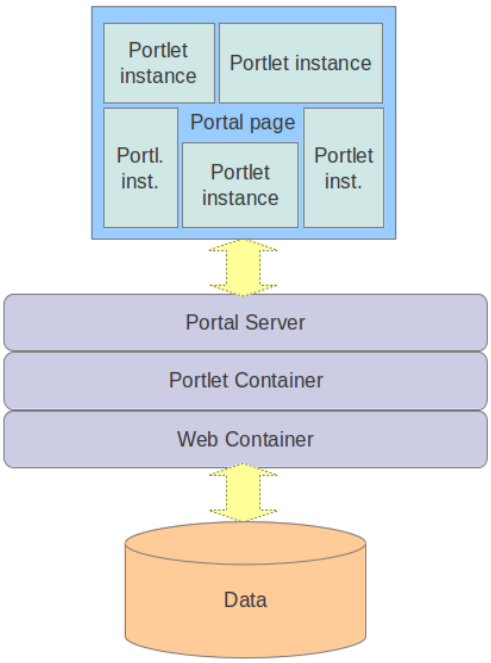

Figure 9: Illustration of portlet infrastructure.

above).

Portlet instances

The instances of portlets are created by the portlet container invoking the life cycle method on the portlet. The portlet instance is then managed by the portlet container.

3.4.2 Portlet life cycle

If we put it all together we can describe the life cycle of a portlet. The portal server and portlet container roles in handling a portlet request. The portlet container creates the portlet instance and handles over the content to the portal server. The portal server then assembles the content from different portlets to generate the portal page.[1]

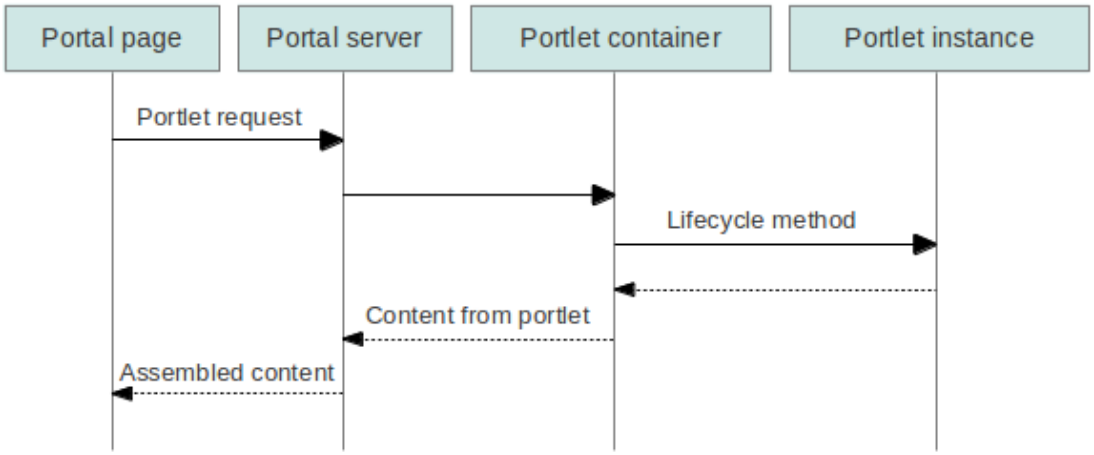

Figure 10: Portal server and portlet container roles in handling of a portlet request.[1]

Now we can look on more illustrative scheme of portal page generation respecting the presented life cycle. On the figure bellow we can see a bit simplified process of portal page generation from an instantiation to a final result shown that can be seen in a browser.

Portlets A, B, C, D and E are generated and its contents are sent to the portlet container. Container then handles them to a portal server that have to integrate them within the portal page respecting defined layout. On the portal page are displayed windows A, B, C, D and E, each for corresponding portlet.[9]

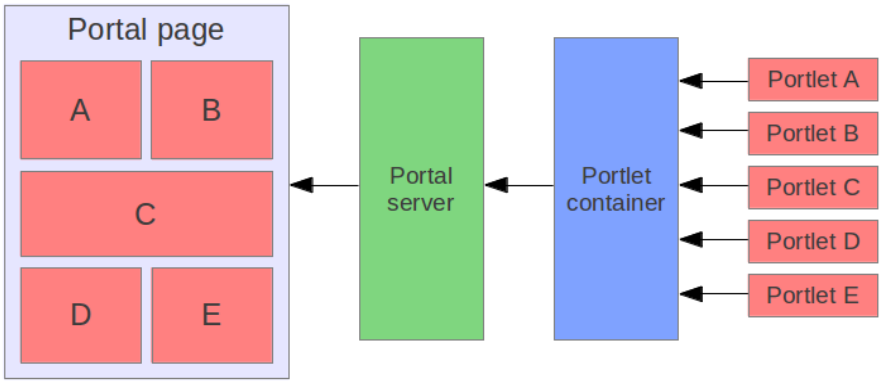

Figure 11: Portal page generation as described in paragraph above.[9]

3.4.3 Portlet modes

Portlet specifications provides some predefined modes, in which the portlets could be rendered. This modes defines the use and functionalities the portlets have when rendered.

Specification of the portlet modes

According to the specifications JSR-168 and JSR-286 portlet mode indicates the function portlet is performing in the render method. It advices the portlet what task it should perform and what content it should generate.

From the specification there can be found three predefined rendering modes:

View. Generates markup reflecting the current state of the portlet.

Edit. Allows a user to customize the behavior of the portlet.

Help. Provides information to the user as how to use the portlet.

When invoking a portlet, the portlet container provides the current portlet mode to the portlet. However, portlets can programmaticaly change their mode when processing an action request.

The availability of the portlet modes for a portlet can be restricted to specific user roles by the portal.

3.4.4 Window states

It has been defined that portlets aggregates within the one portal page, but there was not mentioned anything about the amount of space in the portal page that portlet occupies. However, the specifications defines a three types of window states and manner how the portlets shows and behaves on a page. There are:

Normal. Normal mode indicates that a portlet will be shown in a standard view - it will share the page

with the other portlets.

Maximized. This state declares that the portlet is only one rendered in the page, or it has more space compared to the other portlets in the page.

Minimized. In this state the portlet should only render minimal output or even no output at all, nevertheless, there can be defined custom states for the specific purpose.

3.4.5 Portlet deployment descriptor

Similarly, as for a classical JEE web application there is a deployment descriptor that roles as a form of a configuration file providing informations used by the application server during deployment of the application to the web container, in a portlet application there is a portlet descriptor that holds the informations and settings used by the portlet server during deployment to the portlet container.

Deployment descriptor for a portlet application is an XML structured file named *portlet.xml*. Next we will look into descriptor and describe interesting parts of its structure.

Portlet descriptor structure

The root element of the file is <*portlet-app*> element. Portlet declaration is enclosed within the <*portlet*> subelement of the <*portlet-app*>. There can be even more portlets declarations within one descriptor. Portlet application will then consists of that many portlets as specified in the descriptor - all these portlets form a part of one portlet application.[1]

Within the portlet.xml it is a place for declaration of portlet name, used portlet class, enabled portlet modes or window states. However, there can be set a list of parameters and options, they will be presented on demand later in the text. For example, the simple portlet.xml can look like this:

```
<portlet-app>
  <portlet>
    <portlet-name>myPortlet</portlet-name>
    <display-name>My Portlet</display-name>
    <portlet-class>org.portlet.MyPortlet</portlet-class>
    <init-param>
      <name>email</name>
      <value>my@mail.com</value>
    </init-param>
  </portlet>
  <container-runtime-option>
```

```
      <name>javax.portlet.actionScopedRequestAttributes</name>
      <value>true</value>
    </container-runtime-option>
</portlet-app>
```

Figure 12: Example of a portlet.xml file.

3.5 Inter-portlet communication

Inter-portlet communication is a process that provides an ability of a portlet to react on changes, events and actions which take place in the other portlets. An example could be a portal of library where the reader hove shown within the account portal page both the portlet with the list of available books and second portlet of his own reservations. When the reader makes a reservation for a book through a reservation portlet, it is wanted that this reservation instantly occurs in the portlet showing actual reservation list. This can be done by an inter-portlet communication mechanism. Let's describe its abilities and functionality.

There are three mechanisms that can be used for the inter-portlet communication:

Portlet session. Portlets are parts of the same portlet application or the portal page enables sharing of session information over the different portlet application within the portal.

Public render parameters. Communication based on the simple string values sending between the different portlet applications.

Portlet events. When portlets from the different portlet applications are communicating with each other by sending and receiving complex objects, known as *events.*

3.5.1 Portlet session

This is the most commonly used type of communication between the portlets. It is the one already exists since the JSR-168 (Portlet 1.0 specification). The main component is a *PortletSession* object storing, maintaining and providing an access to session attributes. As it was mentioned earlier, there are two different contexts, in which the session data can be stored and which defines and limitate the scopes of access to the attributes stored in:

APPLICATION_SCOPE. It is used to store data that need to be accessible within the whole portlet application so the portlets of the same portlet application can access them.

PORTLET_SCOPE. This scope provides session data only for use within and for the current portlet instance. So the the portlet session data cannot be shared even by the duplicate instance of the same portlet application on the portal page. However, we are not unable to retrieve this data from the other

portlets, as the data are also stored in the application scope, although in a different way, so we can programmaticaly make an access to them.

3.5.2 Public render parameters

When the render method is fired, the parameters that are required by the portlet to generate the appropriate content are handled to it. These are called *render parameters* and there are two types of them:

Private render parameters. In the requests that are not visible to other portlets.

Public render parameters. In the requests that are visible to other portlets within a portal page.

The render parameters are received by a portlet in the situations, such as a form submission to a portlet's render URL. In this situation the form fields are sent as render parameters to the render method. Although, submitting of a form to the render URL, however, it changes the system state, it's not recommended in the render phase. In the same way are sent the parameters set in the portlet's render URL that references the render URL. Finally there can be render parameters sent to the portlet through the render method call, if they were set in the *ActionResponse* before.

These parameters are implicitly private. So when there are sent request parameters as in the previously listed, there are always automatically treat as private. And as private they are visible only for a current portlet.

If we want to create public request parameters, and therefore visible for the other portlets in the application, we must also specify the names of the request parameters we want to make public in the *portlet deployment descriptor* (we will look in a more detail within the applications in the phase of analyze). Thus the scope the container knows that the request parameter must be changed in order to be visible to the other portlets. The public render parameters are additionally available in action, resource and event methods.

3.5.3 Portlet events

The *portlet events* were introduced in the JSR-286 (Portlet 2.0 specification). On oppose to the previously described mechanisms for inter-portlet communication, the with events the sending portlet is not responsible for the delivery of the event to the receiver portlet. This responsibility is put on the container. Let's clarify this:

As there were mentioned, we can call the two sides of inter-portlet communication: *sender portlet* (a portlet that invokes or sends event) and *receiver portlets* (the portlets that listenes or states they want to receive this event). When sender portlet fire the event, the event is sent to the portlet container. The container is then responsible for marking the event available to receiver portlet. Portlet

container knows the receiver portlet and it handles the event to them as shown on the figure bellow.

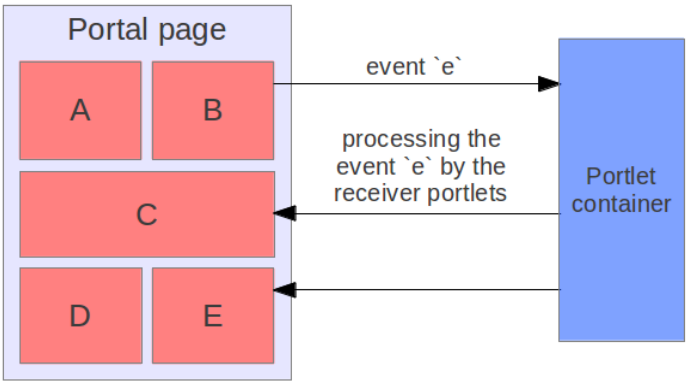

Figure 13: Sender portlet B generates an event and sends it to portlet container, event is then processed by the container to receiver portlets C and E.

Advantages. The advantage of communication through the events is that the event can transmit a complex information on oppose to the public render parameters, where you can change only String object information. There is also an ability of event to fire another event from itself, and so there's possible to create even a more complicated chains of events.

Disadvantages. On the other side a disadvantage of events are that they are a bit clumpy on both sides: container and code increase. For the use when need to send and receive a primitive data it is better to use public render parameters.

3.6 Portals and portlet frameworks

The platform for the portlets is the portlet container that is usually included in the frameworks providing portal deveoping tools. They also provides *Content Management Systems* (CMS) to bring developer more comfort when building a portal. There are number of commercial technologies coming from the application server vendors as the IBM (WbeSphere Portal), BEA (BEA Portal) or Oracle Corporation (Oracle Portal). And there are also number of open-source and free technologies that provides not fewer functionalities.

The most popular from the currently existing open source technologies is the project of the Liferay, Inc., called Liferay Portal. It is also available with a packed Apache Tomcat web container in it and provides a simple CMS to build a portal. Many books encourages the beginners in the portal platform to use this technology and brings a different tutorials that helps to build up a portal without almost any requirements.

Second, well known open source portal is GateIn Portal, that comes merges the two mature projects: JBoss Portal, provided by RedHat inc. and Exo Platform developed by the company eXo.

However, the GateIn Portal will be used as implementation technology for the project, we will provide small introduction to the technology in the next section.

3.6.1 GateIn Portal

GateIn Portal is one of the key components in the *JBoss Enterprise Portal Platform* (JBoss EPP). It provides web portal and also a portal framework to build upon. As the follower of the JBoss Portal it also includes the *JBoss Portlet Bridge* that is an implementation of *JSR-301 (specification of Portlet 1.0 Bridge for JavaServer™ Faces 1.2)* and *JSR-329 (Portlet 2.0 Bridge for JavaServer™ Faces 1.2)*[3]*;* they also supports JBoss web frameworks such as JBossSeam and RichFaces.

Web Services for Remote Portlets (WSRP). Among the other GateIn Portal technologies integrated in, there is also a technology called *Web Services for Remote Portlets*. The aim of WSRP specification is to define a way how to display remotely-running portlets inside the localy-running portal page. This is enabled by leveraging the *SOAP (Simple Object Access Protocol)* and use of the set of predefined common interfaces. Interaction is then performed through the SOAP messages.

3 The specifications which defines semantics for executing *JavaServer Faces (JSF)* views within a portlets

4 Etherpad Lite and EasySync

The two main pillars on which the project of a collaborative editor as a portlet will be built are Etherpad's EasySync library and the referential implementation leveraging the Etherpad libraries, known as Etherpad Lite.

4.1 EasySync protocol for collaborative editing

Etherpad uses a protocol, that encapsulates change made on a document by single writer into small and lightly transmittable form. Although, we are not focusing on reimplementing this protocol, we will be close enough to that mechanism; therefore it will be usefull to introduce its main principle a bit.

4.1.1 Overview

The EasySync protocol transmits a changeset as a pair of entities; these are Changeset and Attribute Pool.

Changeset. Although, we have already described term changeset in common, in this case we will talk about the specific implementation. Our Changeset is a string-encoded representation of change made within the document - it carries an information about the most recent changes within the text. Such informations are: amount of increase or decrease of text amount, whether it is an addition, removal or change of characters; even a change of its style and the location of this changes. When addition is submitted, the Changeset holds also the added characters.

AttributePool. Attribute pool is a collection of attributes used within the Changeset. It is a map of attributes which acts in a current change of text. An attribute is represented as a tuple of attribute name and its value. These attribute definitions are within the map identified by the number key which is also referenced from the Changeset string. The Changeset string also defines where it will be applied. The Attribute Pool also carries an information about the writer that made the change and the timestamp when the change was made.

4.1.2 Changeset structure

As it was told, our changeset is a string representation of the most recent change made on the document. It consists of alphanumeric characters and special reserved characters with defined meanings. Let's show an example of such string for a more illustrative description; we have this Changeset string:

```
Z:9y>1|8=79=2n*1+1$a
```

The string can be divided onto several main parts; however, we will describe step by step the meaning

of current values in respect of order:

`Z` *magic character*, it is the first character of the string and it retain practically unchanged, as it denotes the version of protocol currently used,

`:9y` this part tells how long (in base 36) was the original text (9y ~ 358 characters),

`>1` the manner of change - whether it was extended (>), retained (=), or decreased (<) and how (>1 ~ increased by (1_{36}) character),

`|8=79` denotes how many characters will be retained (=), removed (+) or appended (-) involving newlines, for us (79_{36}) characters will be let unchanged and (8_{36}) from them are newlines,

`=2n` without the pipe sign (|) the notation says, that the newline characters would not be involved, so it specifies that number of following characters ($2n_{36}$) will be retained (=), removed (-) or inserted (+) (none of them newline),

`*1` here we have, mentioned before, reference to the attribute transmitted over in the Attribute Pool, the number starts at 1 and it is related only to the actual Attribute Pool, the attribute declaration tells, that followed slice of text would apply this attribute,

`+1` this means an insertion (+), removal (-) or retention (=) of the specified number of characters (1_{36}), however the previous declaration specified the attribute, this attribute would apply to this (1_{36}) characters,

`$` this character is a delimiting sign that divides the changeset string into two parts, the declarative part and the stack of characters (if any inserted),

`a` this is the last part of the changeset string consisting of a list of characters inserted into the text within the current change.

Now let's move to the description of the AttributePool's structure.

4.1.3 AttributePool structure

Attributes within the AttributePool are formed as a JSON-friendly array. The example could look as following:

```
{"0":["author","a.5pXEmWfWEIT9cuyv"],
 "1":["underline","true"]}
```

On the example shown above we can identify the array of two items, each with numeric *key* and a

value, which is a tuple. The key is value that was referenced in the changeset after the *-sign. As was said in the description of changeset string, it tells that the attribute will be applied on the following (inserted or retained) character specified in shown example by *+1*. If we put these facts together we can tell, that the inserted character *a* will have the attribute *1* applied on it. With a glance on the AttributePool, we find out that the number *1* key is assigned to the atribute with name *underline* and its value is *true*. The *underline* attribute could have only two values, telling whether it is underlined or not. Therefore the current value means, that the character *a* will be after application of changeset underlined.

What we have not mentioned yet is the attribute *autor*. As you can see it precedes the underline attribute, although it is not presented in the changeset string. The author attribute denotes the identification of user, that made a current change and as from the current session could not came a change made by another user, it is useless to involve it; therefore the attribute is provided only within the AttributePool.

4.2 Etherpad Lite implementation

As was already introduced, Etherpad Lite is an open-source project built with a motivation to bring more light implementation of the project Etherpad. It leverages its EasySync library and JavaScript version of editor used within Etherpad. It also brings a new management of pads and adding an interactive chat placed on the pad editor. The project is programmed in JavaScript and built upon a JavaScript library NodeJS to model the server-side of application.

Although, an original Etherpad is open source project too, it is more complex to compare to its follower. It is also made with use of different technologies on oppose to clearly JavaScript way that took the Etherpad Lite. The third argument why to prefer study the later project is that, as it was told, its practically Etherpad's upgrade and latest referenced revision.

4.2.1 Server

Although, JavaScript is widely known as a client-side programming language, it can be used also on the side of the server. In the Etherpad Lite project it is provided with the use of the following frameworks:

NodeJS. Event-driven I/O server-side JavaScript framework built on V8 JavaScript engine from Google and several other libraries.

Socket.io. Its API provides bidirectional communication over the web. It is lightweight and therefore used for a client-server communication between the user pad and the server app.

UeberDB. Database abstraction layer turning the database into a simple key value store, at the

moment. Currently, supporting only MySQL and SQLite. ueberDB uses a smart cache and buffer algorithm to provide faster communication with the datastore. The optimalization is done also on the side of the read/write operations. However, the reads are cached and writes are done in a bulk, the overhead of database transactions is reduced.

Async. Utility module that provides functions for working with asynchronous JavaScript. As it was originally designed for use with node.js, it can also be used directly in the browser. Async provides functions as map, reduce, filter, forEach… as well as some common patterns for asynchronous flow control (parallel, series, waterfall…). The functions are close tied with the node.js, so it must be followed the node.js convention of providing a single callback as the last argument of your async function.

Express. Lightweight and fast JavaScript server-side web development framework built on Node.js and Connect (middleware layer library for Node.js).

UglifyJS. This toolkit provides functions of parsing JavaScript code, compressing it, or even transforming it to nicer code. Within the project it is used mainly for minimization of code and therefore minimizing the overhead.

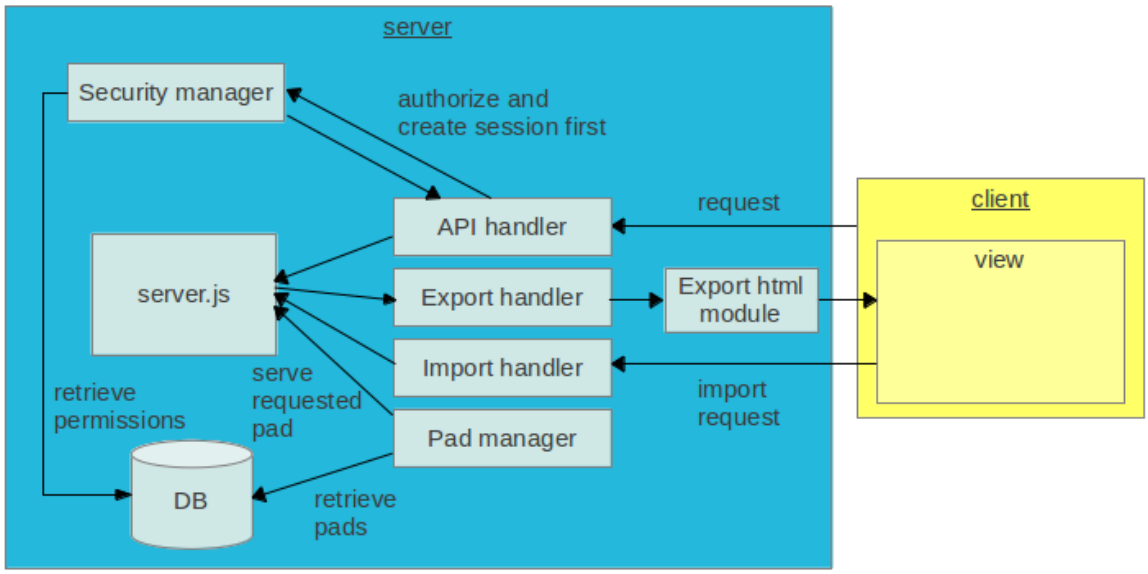

Figure 14: Brief illustration of client-server communication within the Etherpad Lite infrastructure.

4.2.2 ACE editor

ACE states for *a collaborative editor* - a project providing functionality for a real-time collaboration on editing the documents.

ACE editor is already leveraged by the both of projects, we are discussing. The Etherpad Lite, however, a bit overrides the existing API by providing tight integration with the management layer that encapsulates whole pad functionality of the ACE editor.

4.2.3 Pad management

The concrete pad is managed by the pad manager object, that is responsible for establishing and maintaining the connection with the server, retrieving the pad, dispatching the actions within the editor and delegating them over to the server. It also provides notifiers for the actions held within the remote collaboration user editor to be able to update the state of local editor with that of collaboration user.

However, the JavaScript environment does not provide session management - there are also involved subelements which are responsible for all the aspects of the concrete pad:

ACE2Editor. Main module of the pad, holding the ACE editor, that provides the HTML frame representing the editor. All the needed functions which can be called by pad manager and all the events, which may take a place within the editor are registered through this module.

PadEditor. The pad editor must prepare the editor before it is initialized, set view options of the editor, restore a revision text and even disabling or disposal of the current editor.

PadEditBar. It is responsible for maintaining the editor toolbar state and actions made on it.

PadDocBar. Manages the configuration of the toolbars and pad.

PadSavedRevs. Manages the stored revisions of the current pad.

PadUserList. Manages the list of users connected to the current pad.

PadConnectionStatus. Responsible for maintaining the current connection status of user.

PadCookie. Manages the retrieving and storing the cookies for the user session.

PadImpExp. Management of the import and export actions revoked from the pad toolbar.

PadUtils. Auxilliary utilities providing the date formatting, URL escaping and other frequently used functions.

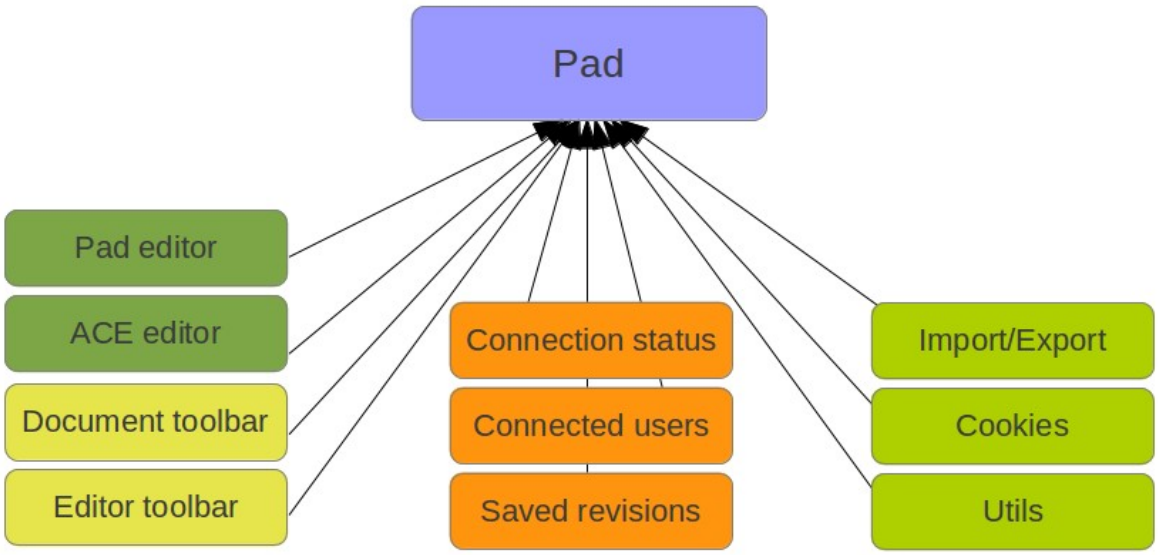

Figure 15: The illustration of Pad manager decomposition.

4.2.4 Data storage

Persistent data of the pads can be stored in Etherpad Lite either in DB or in a file, since used technology provides support for both the mysql and the sqlite databases. However, in both cases the data are not normalized at all. As it was mentioned before, the ueberDB data storage stores different records in a form of a map consisting of two columns: *key* and *value*; diverse data is therefore stored in one database table.

Description of table columns in ueberDB:

Key. This is a primary key column although, it has not ordinal value. Value is provided as a string concatenating attributes, that are tied with this record. The value could therefore rise fast to the high lengths. Nevertheless, the algorithm is obviously aware of it, as the declared maximal available length of key value is 100 characters.

Value. Value is a *longtext*-typed string in a form of dictionary and it comprises every attribute significant for current action.

5 Persistence tier design

5.1 Domain analysis

For the sake of clarity of the integration process: the presented meta model will be relatively simple providing no advanced functions. The model will try to cover just the main objectives of such an application until providing working integration with a portlet technology.

5.1.1 Entities

There were identified following entities in the implementation of Etherpad:

User. User is a main actor in an application of a collaborative editor. Each user must have unique identification within the application, which can be either identification number or an unique user name. Thanks to Single Sign-on mechanism provided by portals, we do not need to store any additional passwords for users.

Pad. Pad entity stays for a document, that can be created, modified or deleted. The attributes should include unique identification that can be same as within the User with a restriction on uniqueness of the names.

Changeset. An essential part of the CE and revision systems s Changeset. It carries changes made on the Pad. Changeset must includes the changeset string encapsulating the change on the Pad with a use of the EasySync protocol.

AttributePool. As it was already mentioned, the AttributePool is co-transmitter of Pad modification, including additional informations for the Changeset.

These are the base entities for modeling of a collaborative editor system. After small introduction to domain we will look on the relations in it next.

5.1.2 Relations

To provide a basis for modeling the persistence logic we need to describe also relations between the entities above. Following class diagram denotes the relationships between the previously described entities:

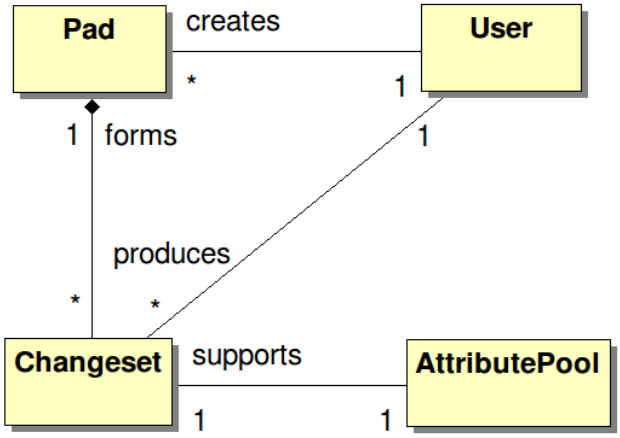

Figure 16: Class diagram denoting core domain objects and relations.

The important entity within a CE model is a Pad. It is a document, that can and must be created by an User. However, who is the creator of Pad is not an essential information of core CE functionality, we should not omit it.

The key to the less importance of relation between the Pad and User is in the next relationship. It is that between the User and Changeset objects. Changeset is the most significant part of the editing process since it reflects a change made on a document (in a Pad) and it is made by an User. It can be therefore created only by the single User. However, the Changesets bears the whole information about the changes made on the document, it is not needed to provide a Pad with such informations.

As a previous figure denotes, Pad object is formed from the Changeset objects.

The third relation drawn puts together Changeset and AttributePool. Changeset will carry the string rule defining the current change to a document. If we were building a very simple text editor, where there is only one available color and no special listing signs and only one style for fonts, we will last with only Changeset object. It is due to the fact, that the AttributePool carries only such attributes that needs to be specified by a tuple (key-value). Therefore the decomposition is better, as be could let open the other options. The AttributePool instance should be, same as a Changeset, unique and the relation must have cardinality 1:1. However, as we can provide clear decoupling on these two objects, the relation could be made more effective. We will discuss this issue later in this chapter.

5.2 Extending the meta model

In this topic we will move the analysis to the classes in the application.

5.2.1 Temporal entities

Some entities described in the previous topics have a character of temporal entities. We have the Pad entity, which as a standard document should provide informations about its creation and modification time. Although, the creation attribute have to be provided within the Pad, the behavior of the changes made on it brings better solution for the modification time property.

This take us little bit aside of persistence layer, since we must have a look at the Changeset entity. As we know, every change made on the Pad has a reflection in form of Changeset object. Therefore, if we provide this entity with a creation time attribute – same as within a Pad, we can reuse its value also as an information about the time of change on the Pad. Discussed idea is shown below on the next figure.

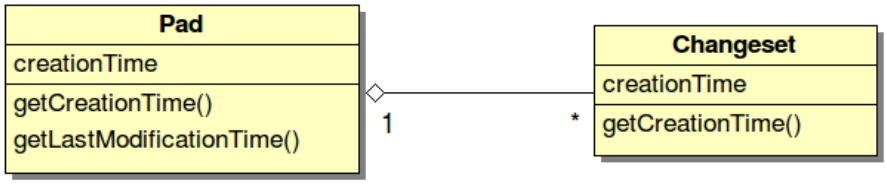

Figure 17: Class diagram denoting Pads and Changesets properties providing temporal informations.

According to the previous figure, the *getCreationTime()*[4] method has in both cases role of *getter* method, providing the applicable private property *creationTime*. The second method denoted within a Pad class - *getLastModificationTime()* method would provide the desired value revoking the method *getCreationTime()* on the latest Pad Changeset object. The method getLastModificationTime() can look like this (for the sake of simplicity: we are supposing that the appropriate methods are defined and members presented; also we suppose the list of changesets is not empty and method getCreationTime provides legitimate value):

```
public long getLastModificationTime() {
    Changeset latest = this.getLatestChangeset();
    return latest.getCreationTime();
}
```

4 Some of the property names may not match with the implemention for sake of description.

5.2.2 Attributes

Behavior of the application puts a big requirements on the DB resources. If we will build on the base meta model shown previously in Figure X., the CE could provide dozens of new records during the user work. This amount would be even raised by the number of users writing. Therefore, the model should be optimalized in order to provide more effective persistence logic. We will discuss two topics, depending on the place of the optimalization, where will be provided.

Relationship between Changeset and AttributePool

When the user is writing the text into Pad, he probably will not change the attributes for every single character he writes. The assumption is even on opose to it, however, the written text would very likely comprise of continues text chunks, which all have the same attributes. Therefore, we should decrease the amount of the redundant AttributePools changing the relationship to Many-to-One. Simply, different continuously submitted Changesets could share the same AttributePools.

Updated class diagram will look like following:

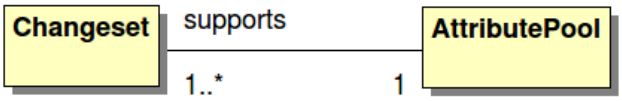

Figure 18: Relationship between Changeset and AttributePool with a new cardinality.

AttributePool

AttributePool object will comprise a set of attributes and their values to perform the additional configuration for the current Changeset. Although, the attributes within the AttributePool can differ from instance to instance, there is a certain quantity for both the attributes and the applicable values. Since storing every attribute with a full name and value could obviously provide also bit of redundancy, what can be relatively expensive for some aspects:

Storage capacity. The frequency of changes would raise to high values and therefore produce a huge amount of Changeset and AttributePool objects, that must be stored within the database.

Database operations overhead. The previous aspect also impacts time that the database operations lasts.

However, the values could be also extracted because there is a limited number of them (for each attribute), the occurrences probably would not be as frequent as in the case of attributes; therefore, we could retain them for a now with, maybe, a bit benevolent behavior. Discussed solution is also denoted on the next figure.

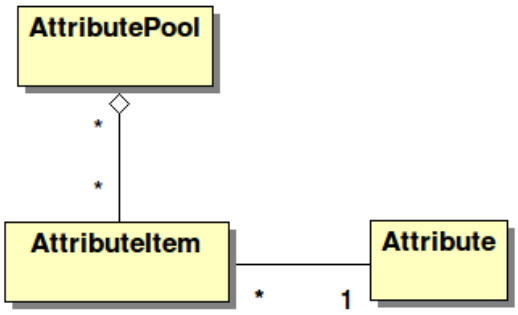

Figure 19: Class diagram describing the AttributePool entity breakdown as a aggregation of more AttributeItem objects.

5.2.3 User session configuration

When User signs into already visited Pad, he should be provided with previously specified preferences. These preferences could be: text color, information whether to show numbers of lines or not and other editor settings that could be specific for the each User. A simple Session relationship is illustrated on the next figure also providing time informations keeping the user evidence.

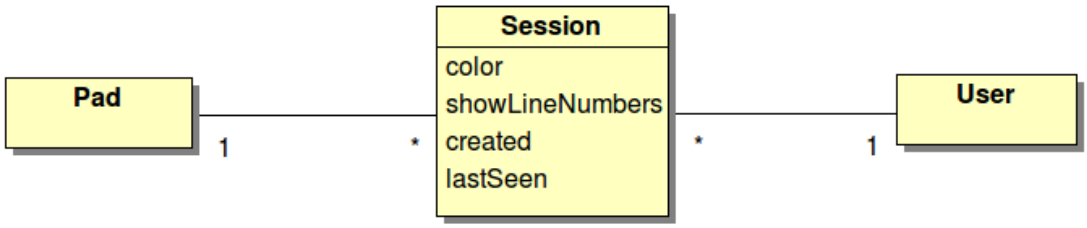

Figure 20: User signed into a Pad creates a Session to provide informations specific for the current relationship.

There will be a small relatively static number of preferences, User could specify; therefore we could store them within the Session entity in form of attributes. The change made on available preferences such as e.g. addition of a new preference will require the update of the Session entity. This would not be problem for the database table, but it could bring complications when updating the application logic. Thus, we could provide a little different model, that would be more flexible. This approach is similar to that we made for the attributes within the AttributePool. Next figure shows such model.

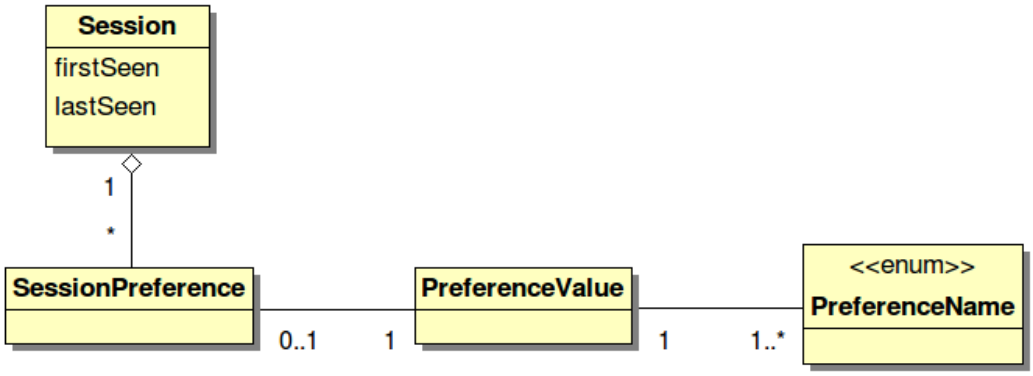

Figure 21: More sophisticated approach of modeling the preferences in the Session relation.

This model of preferences if compared to that of attributes also provides us with specified values which is preferred, as it assures using only regular ones. Additionally, advantage of this approach is that it decreases amount of data, as the preference values are only referenced, not duplicated for every SessionPreference item. Since there will not be many session attributes and since it is also not the most essential part of the CE, it is not needed to provide such a decomposition and we can stay with the first diagram.

5.2.4 Final meta model

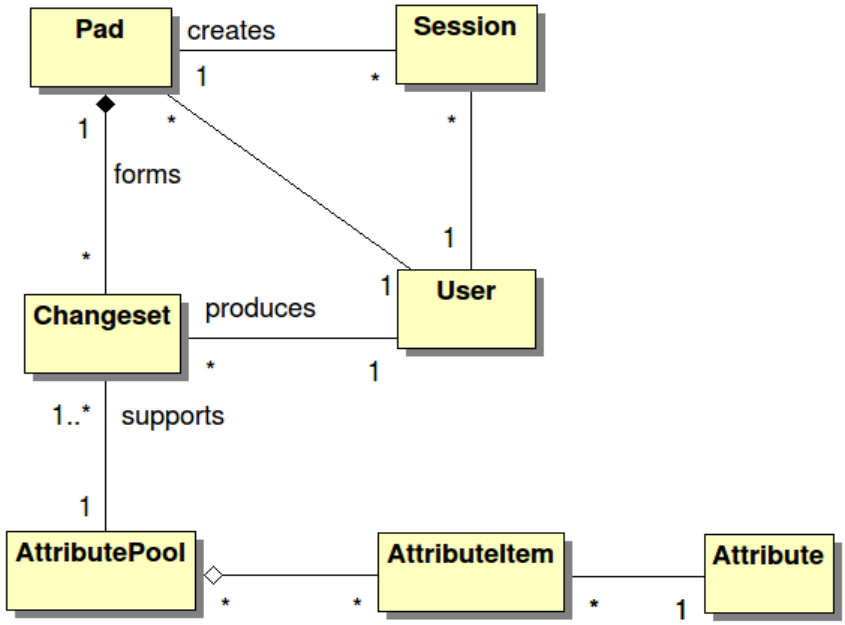

Figure 22: Full meta model for our CE implementation.

In the figure above we can see whole meta model of our application as it was discussed in the previous sections. The following section will build on this meta model to provide the model of database.

5.3 Design of the database

As the first thing in the development, we need to build up a correct database model, since it is a base module of application and it can define even the simplicity or complexness of application logic needed. We have also created a meta model that will very likely fits the actual database model; therefore we will describe the model by looking on the single tables and their attributes.

5.3.1 Tables

To reduce the amount of text and to highlight more interesting parts of the structure we will extract some attributes. They are:

Common attributes

Id. All of the mentioned entities will be provided with the attribute *id,* which will be used as an identification of a record and *primary key* for the table. Definition of the column will be in all cases the same and in DDL syntax for MySQL should look like this:

```
`id` integer(10) NOT NULL AUTO_INCREMENT,
primary key(`id`)
```

CreationTime. This attribute is also common for more than one table. It bears the information about the time, when the entity object was created. As default value current timestamp on record creation will be used. The definition of the column will look as following, respecting the syntax of DDL for MySQL:

```
`created` timestamp DEFAULT NOW()
```

User

Within the portlet application we do not need to provide an authentication and authorization of users since it is already provided by the portal. Although, all the informations about the users are held in the portal, we can provide our User with just a few informations retrieved from the identity management system of the concrete portal. We will specifying the name of user, that will be already loaded from the portal and optional extern User id, that will hold the id of a User in a portal.

Pad

Besides mentioned attributes, Pad table will provide an information about the name of Pad and author of Pad, that will be also a foreign key referencing the record in User table.

Changeset

Most significant informations provided by Changeset record will be held within the attributes: number,

rule and charbank. Attribute number will hold an information about the order of current changes as they occurred within the Pad. Therefore it is also an information for assembler of Changesets, so it knows the order, in which the Changesets are applied to the parent Pad. Columns rule and charbank will hold the information about the text change respecting the EasySync protocol. The changeset string is, however disjointed into two parts: declaration of change (rule) and the inserted characters (charbank). Foreign keys will be described in the ERD in the following section.

Session

Session table will provide only some attributes specific for the User – Pad relation. For now it will be the information about the color used by User within the current Pad and time when he was last seen in a Pad.

AttributePool

This table roles as a joining table in Many-to-Many relation between the Changeset and AttributeItem. Therefore the only information providing is its primary key by which it is referenced in a Changeset record.

AttributeItem

The table consists besides a primary key column from number column, that references the concrete attribute as it is referenced within the Changeset. Then it holds the reference to Attribute table record to identify the attribute to define its value and finally the value. Every record of this table belongs to one record in an AttributePool, that references with foreign key attribPoolId.

Attribute

Table Attribute holds the available attributes, which can be used by the editor – which distinguishes. It only holds the primary key identification and name identifying the attribute mainly in a browser.

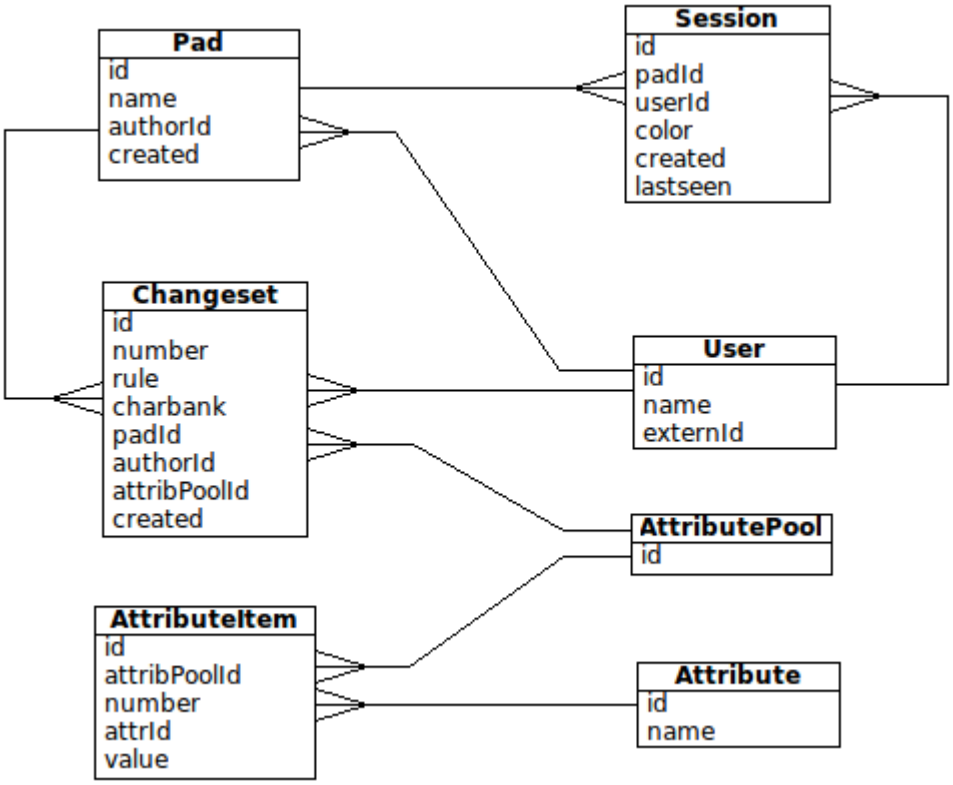

Figure 23: ERD diagram of the database.

All the attributes that was not mentioned can be simply understood by the ERD diagram above. All the tables provides primary keys in a column named as id and all the foreign keys references joining tables with name created as shortening of joining table name or role that can be intuitively held by the current entity suffixing with “Id”. Whole tables definitions can be seen in Appendix A.

5.4 Persistence logic

5.4.1 Hibernate Session

Communication with the database is done through the Hibernate Session object. The Hibernate Session is retrieved from HibernateSessionFactory which uses the configuration provided in hibernate.cfg.xml file to create Hibernate Session. The work is done as follows:

Create the SessionFactory from hibernate.cfg.xml:

```
SessionFactory sf =
      new Configuration().configure().buildSessionFactory();
```

Open the Session on SessionFactory:

```
Session s = sf.openSession();
```

Execute select query:

```
List<Pad> pads = s.createQuery("from Pad").list();
```

Or execute a transaction:

```
s.getTransaction().begin();
s.persist(pad);
s.getTransaction().commit();
```

5.4.2 Data Access Objects (DAO)

DAO Factory

All the manipulation with the database data is done through the Data Access Objects. This design pattern separates the application logic from the database operations by extracting the data manipulation logic into separate objects.

6 Application tier design

6.1 Class model

6.1.1 Base entity decomposition

However, entities described in the previous sections reflects different objects in the application, they have some attributes with the same semantic. These attributes can be extracted into abstract classes to provide more unified definition, its behavior and finally, eliminate redundancy of code. We will describe the extracted super classes and describe its meaning and purpose.

BaseEntity

All the entities we have talked about has one attribute in common. It is *id* that identifies current object. Since the semantics of the id attribute is for all the entities the same, we can extract it into new class that all of them will extends. A class diagram shown below the text describes the inheritance.

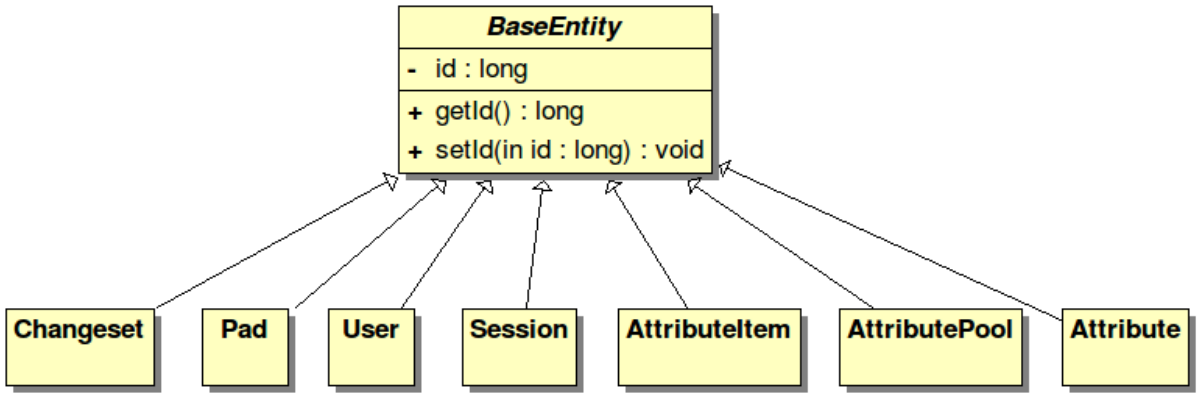

Figure 24: Base entities inheritance – extending an abstract class BaseEntity.

TemporalEntity

We have also some entities which need to provide an information about the creation time and who created them. The touched entities are Pad, Changeset and Session, although, they can be more of them if we wanted to bring wider evidence. The inheritance is shown in the next class diagram.

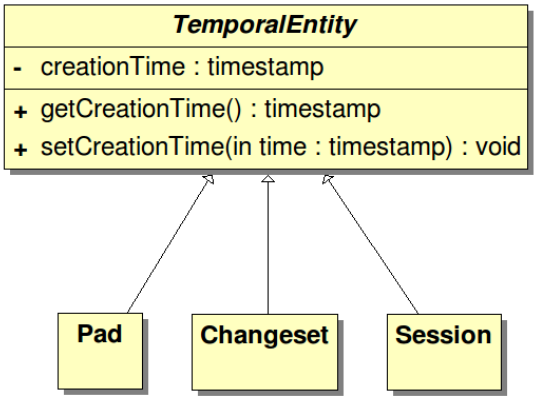

Figure 25: Temporal entities inheritance - extending an abstract class TemporalEntity.

6.1.2 Main relations

Pad – Session – User

On the next class diagram it is shown the both Pad and User class relations with Session class. However, the Session class is just extending of Many-to-Many relation between the two mentioned classes, they both could provide a collection of related Session objects. In our case the collections of Session objects is modeled as a map, where the key value is an *id* attribute of the other one class object involved in relation with Session. For example in the *userSessions* map, the concrete Session object will be referenced by the Pad object id as a key value and vice versa.

The advantage of the model is that we do not need walking through the collection of the Session every

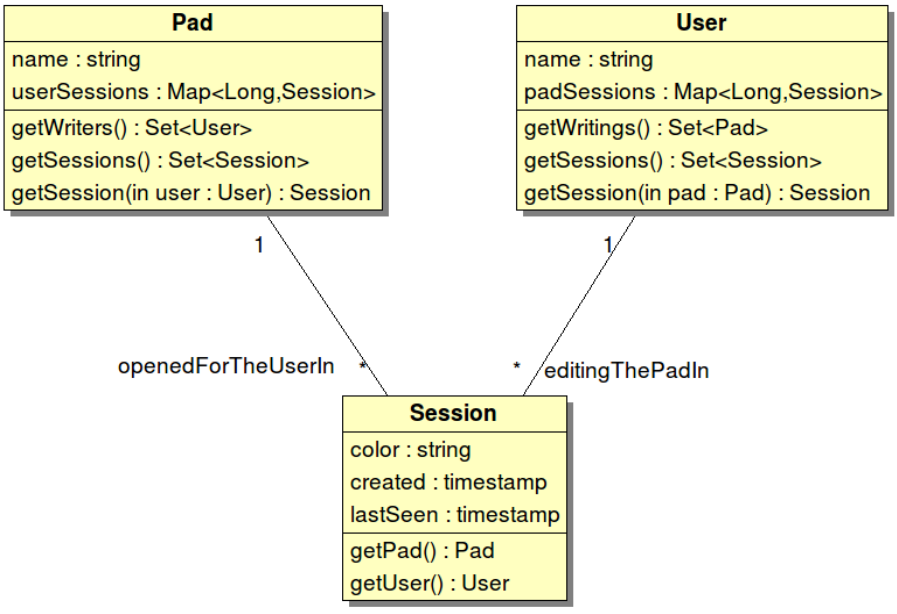

Figure 26: Pad-User relation through the joining Session.

time we want to open the concrete Session.

Pad – Changeset

There has been already told that the Pad object will consists of number of Changesets. However, the Changesets bears only the information about change, not the text all, the concrete Changesets can be applied to a Pad text with the exactly the same state in which the Pad was when the Changeset was made. The Pad therefore must apply the Changesets in a specified order, nevertheless, no Changeset may miss, as it would be impossible to apply the next ones. Thus, the Pad will comprise an ordered set of Changeset objects.

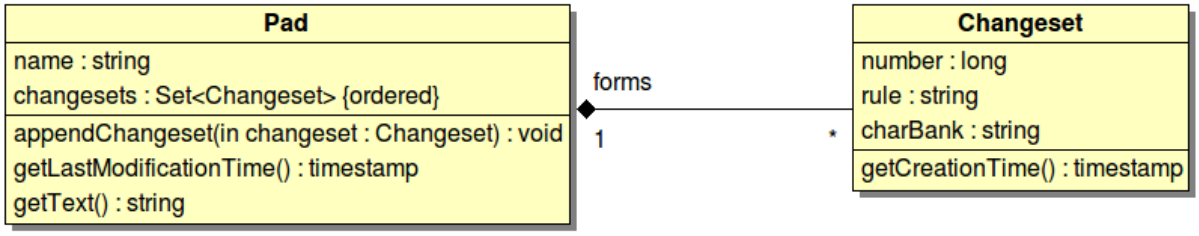

Figure 27: Pad-Changeset relation – composition of Changesets.

6.2 Processing

6.2.1 Client-to-Client communication

The behavior of the CE puts the client application to need of synchronizing the actual state with the other users in the real time. Communication based on storing and retrieving every change into database would cost very much. Maintaining the communication through the sockets on the otherside could bring cost of the connection management and building such infrastructure.

In the modern enterprise application servers there can be found a mechanism called Java Message Service (JMS), that works as a mediator between the clients in the application server. There are two types of mediators defined in the JMS: queues and topics.

Queue. Queue provides point-to-point communication. There are two sides in the communication: the

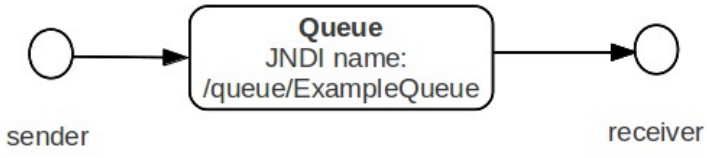

Figure 28: Communication through JMS Queue.

sender and the receiver. The message sent by the sender can be retrieved only by the one defined

receiver. Although, the name seems to predicts the behavior, the order of messages would not be necessary kept.

Topic. On the other side, the topic represents communication in a way of ether. The message is published also by one sender (publisher), however, it can be received by the more listeners (subscribers).

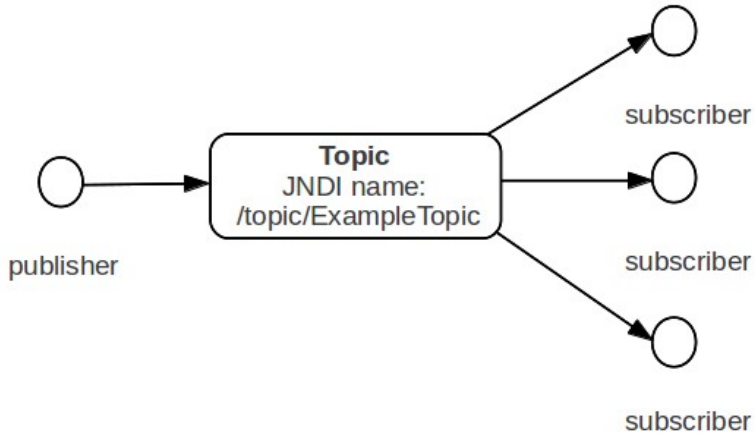

Figure 29: Communication through JMS Topic.

Since in CE application the content in single pad have to be synchronized with all the pad writers, which are editing the same file. Therefore we will choose a topic.

Topic registration

The topics are registered in the JBoss AS through the administration site of started instance (admin-console). Both topics and queues are accessible through the JNDI[5] lookup.

6.2.2 Handling local text change

When the user modifies the document locally in his editor, the easysync's JavaScript function is called for packing the change into changeset string to produce the changeset for the application. Returned changeset is then propagated to valueChangeListener method of the ManagedBean. Then the changeset string is parsed into Changeset object, that is appended to the Pad of current Session.

Publishing the Changeset

The second step after the change is locally handled, the Changeset must be published to the topic, so the collaborating editors could synchronize their states with the modified one. For the sake of simplicity we will for now suppose that two or more Changesets with the same base[6] would not met.

5 Java Naming and Directory Interface – service provided by JEE app. servers for retrieving server resources

6 Base states for the state of document before application of the Changeset.

6.2.3 Handling remote text change

Subscribing to the topic

To be able to watch and react to remote changes, the local Session need to register Listener which will be watching the changes in remote Sessions. The listener is connected to the concrete topic and hear for the events that may occur on it. When the event occurs the *onMessage* method is been triggered with the *TextMessage* object as a parameter. This object holds the message from the sender, The receiver would then parse the message into Changeset and applies it to the Pad.

6.2.4 Session Sender and Session Listener

Every user editing the Pad is doing it in distinct Session. Thus the Session object is one that represents the instance of the editor with the Pad concrete opened in it. Therefore the Session object should be both the Listener and Sender of the changes. However, the Session could not extend more than one interface, this objects will be provided as Session properties.

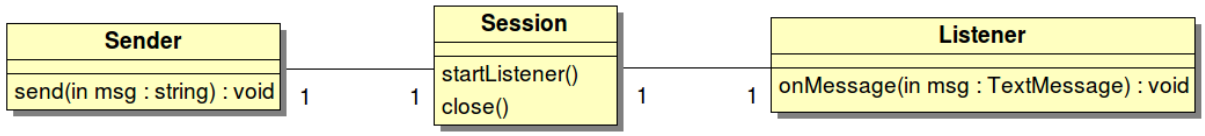

Figure 30: Class diagram of the Session - Sender and Session - Listener relations.

6.3 Authentication and authorization

6.3.1 Single Sign-On

However, identity management (IDM) is already provided by the portal, portlet does not need to provide another one. Since portlet will not be accessible bypassing the portal authentication, the only needed information is identification of the user that can be provided by any information that has a unique value for all the users. For such purpose we could last with such attributes like id in the portal IDM or even login name.

6.3.2 Retrieving user info from the portal

The user credentials can be retrieved from the context above. In portlets there is *PortletContext* class for such uses, that provides a method *getRemoteUser()*. Nevertheless, we can also use *FacesContext* if we are implementing with use of JavaServer Faces technology(JSF), that also provides the method. GetRemoteUser method returns from the application context login name of the User actually logged in, what is all we need for the sake of users distinction. Bellow code example shows the use in JSF.

```
PortletRequest req = (PortletRequest)
                        FacesContext.getCurrentInstance()
```

```
                                        .getExternalContext()
                                              .getRequest();
    String name = req.getRemoteUser();
```

By the user credentials retrieved from the portal the User of the application is searched in the database. If the user was not found – he has not been logged into an application yet, new User object have to be created and stored in the database for the reference of the application.

The responsibility on user retrieving is on UserBean Managed Bean object. First the application start and questions the user info, the UserBean have to provide the User object. The code for UserBean looks like this:

```
public class UserBean {
    private User user;
    public UserBean() {
      loadLoggedUser();
    }
    private void loadLoggedUser() {
          PortletRequest req =
                (PortletRequest)FacesContext
                      .getCurrentInstance()
                            .getExternalContext()
                                  .getRequest();
          String name = req.getRemoteUser();
          user = loadUser(name);
          if (user == null) {
                user = new User();
                user.setName(name);
                save(user);
          }
    }
}
```

Figure 31: Piece of UserBean class code denoting the logic of loading the User.

where method *loadUser* tries to find the User with specified name in the database.

7 Possible extensions

Not all of the ideas was implemented this chapter will introduce or propose some interesting extensions and features that could be later implemented providing better comfort or functionality.

7.1 Import and export

The import and export are both very important features that should take a place in the most number of editors - as it enables mobility of documents and makes the documents transferable and reusable between the many diverse applications.

7.2 Security control and permissions

For the sake of simplicity and due to the portlet API there was not designed any security system. All the security was left out for the direction of portal. Therefore every user logged into a portal will have an access to edit the documents. However, there could be added some kind of system for an access control; such extended tool could be therefore deployed and integrated within the complex structure of users and roles providing for them mandatory access control. We will shortly discuss some of the analyzed models.

7.2.1 Permissions

Entities

The intended security management system would diversify two entities: UserObject and PadSubject.

UserObject. This entity represents the users, it can be although a single user or a group of users. We will say that a single user will be called as the *User* and a group of users as the *UserGroup.*

PadSubject. A PadSubject states for an editable entity. It can be both a document or a group of documents. We will call the document as the *Pad* and a group of documents as the *PadGroup*.

The UserObject is tied with the PadSubject by the type of permission enabling the UserObject to use on it certain functionality. Provided permission types could differ depending on the security model used, however the two main would be the *Read&Write (rw)* and *ReadOnly (ro)* permissions. The next figure shows the relations in the UML notation.

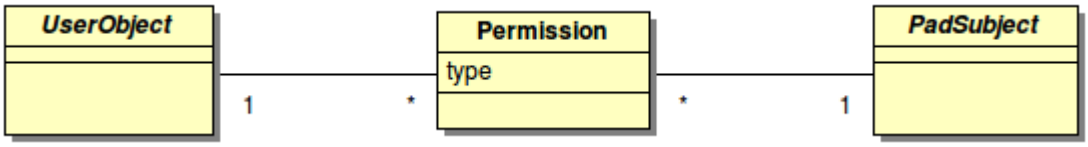

Figure 32: Relationship between the UserObject and PadSubject over the Permission.

Models

There can be designed a number of security models depending on the both UserGroups and PadGroups management manner, however I will present just two attitudes which can be applied on both entities:

Hierarchical. Instances of the entity (UserObject or PadSubject) are organized in a hierarchical structure, where the UserGroup can be a member of another UserGroup and the PadGroup can be a members of another PadGroup. On the bottom level there are the Users and Pads within applicable groups. In such arrangement the permissions can be delegated from the higher nodes down to the bottom nodes. Nevertheless, the closest parent-node's permissions should be applied to the current node. The model is illustrated on the Figure X.

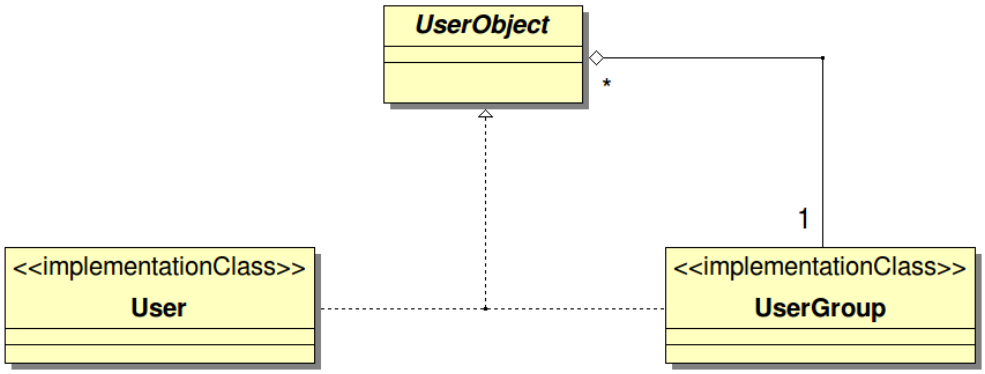

Figure 33: Class diagram describing "Hierarchical" model of UserObject organization.

Flat. Another attitude can be an organization with just one level of aggregation. There are instances which roles as grouping objects (UserGroup and PadGroup) providing the same permissions for its members (User and Pad). There can be modeled two types of membership: exclusive and non-exclusive. In the exclusive model the groups will role as baskets - certain object could reside only within one group. This model, however, will be not very useful for a UserObjects organization, since it disallows dynamic assigning of permissions to the different UserGroups. On the other side the organization of PadSubjects could work in that manner, although it could be rather messy within a large amount of documents. The second model would allow that one object may reside within a more groups which is more flexible compared to the first type. On the figure bellow it is shown the "Flat"

UserObject organization.

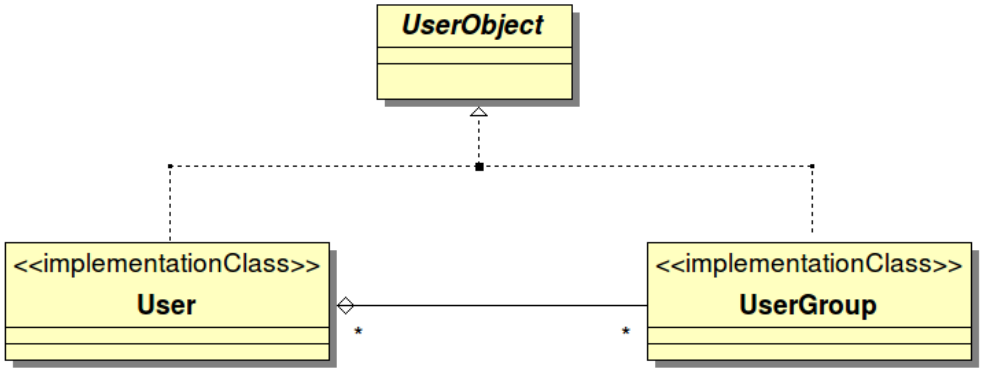

Figure 34: Class diagram describing the "Flat" model of UserObjects organization.

7.2.2 Prioritizing the text – permissions to update or remove

Collaboration on the documents within the group of people can bring the situation when there can be some text or part of the document stated as important and therefore it could be unwanted to remove or modify this slice of text. This can take a place when working on he same document with the collaborating users through the different security levels, so the user on the higher level should have a privilege to put a text that ca not be replaced or removed.

However, this idea also requires addition to the permission types. Although mentioned case suppose the hierarchical model, this idea can be also provided within the flat security model.

7.3 Data optimization and caching

When developing a web-based application that tends to use huge amounts of data stored in a database, you have to deal with the database communication overhead. This is even more critical topic for an application we are developing.

CE places a rather challenging requirements on data management - it is due to high frequency of changes, that could occur in CE and also due to a fact that every change is stored within the separate Changeset (pessimistically with AttributePool for each one) immediately into database. There are two aspects for a database optimization that should be therefore reflected: communication overhead and capacity of storage.

7.3.1 Lazy submission

We have told that the frequency of Changesets generation will be rather big, since the every little change will immediately produce a changeset. This is the issue that can not be overlapped, however, it

emerges from the concept of a collaborative editor.

What we can override is the number of database submissions. However, we have also implemented a protocol that serve as a mediator of changes between the currently working users, the users are already up to date with the current state of actual document. Therefore there is no need to send every change into a database and we can decrease amount of communication with the database to a minimum by providing a lazy submission of changes to a database.

Such a behavior would, however, need a more sophisticated synchronization mechanism in the application logic.

7.3.2 Changesets extraction

The fact that every little change is processed as a Changeset and then submitted into database can cause that its capacity will be very quickly cluttering; therefore a Pad with large content will likely comprise a large number of Changesets. One of the possible solutions for minimizing the number of Changesets within the Pad could be combination of older Changesets into single *initial Changeset* as denotes following picture.

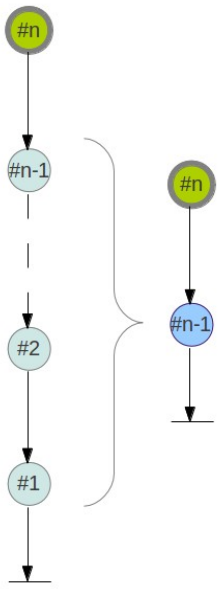

Figure 35: Illustration describing extraction of changesets into one initial changeset.

However, in this case the changeset could comprise a text written by different users and we can not assign an authorship for all of them within a single Changeset object. The solution could be to create

distinct changeset entity for the initial changeset and enable to provide the authorship of every single change in the changeset; this could be provided by the AttributePool. This approach, nevertheless, would not avoid the complications which could be caused just by a different authorship manipulation in the objects of Changeset and initial (aggregated) changeset. For the addition we can look at a class diagram in the following figure denoting discussed idea.

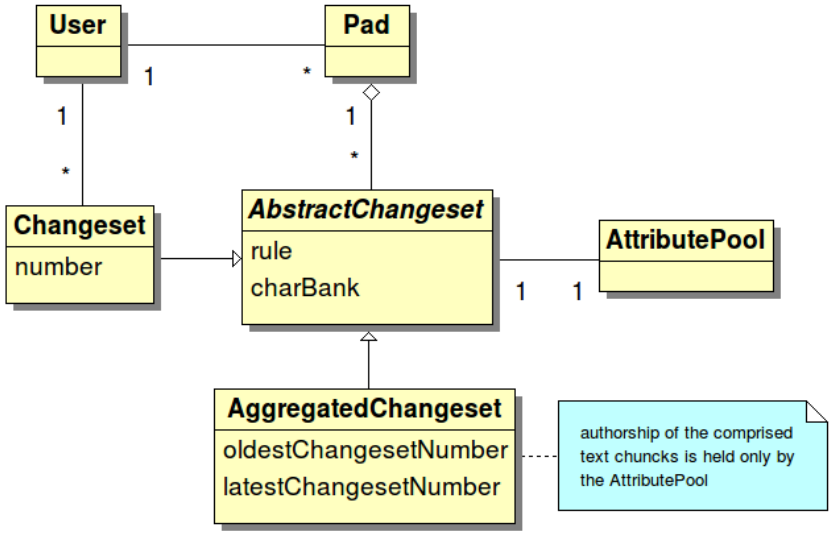

Figure 36: Class diagram introducing a new entity for initial changeset.

On the figure above the AggregatedChangeset states for the initial changeset as discussed, however, the aggregation between the AggregatedChangeset and Pad tells that this changeset can be also placed elsewhere as on the Pad's beginning; as was discussed in the previous paragraph.

This extracted changesets, which where placed in the new AggregatedChangeset object could be then removed from the database and exported possibly into XML file. With this backup, current changeset could be on demand revoked and provided into origin state for use within the application.

7.4 Extended features

There can be provided features that can provide for user some added value or enrich the its functions or usability.

7.4.1 Non-textual objects

For many uses a plain text documents would not sufficiently fulfill users needs. Many times when writing different types of documents, we need a help of figures, diagrams or illustrations; therefore an availability to involve e.g. images could be very helpful. Leveraging enough scalable AttributePool can make the thing very likely. However, there is need of an addition for the database model in a form of a new table that extends the Attribute entity providing the information for the injected text.

8 Conclusion

As can be seen, the work comprises a number of technologies and concepts which can rise much inspiration. Rather interesting theme in the paper is the concept of collaborative editing. The idea brings a big progress into an area of document editing. Collaborative editing can be very useful e.g. in processes of making analysis, decision making or even in extreme programming. Portlet technology itself brings to the field of web development many interesting attitudes; starting with the unification of communication between portlet applications in a portal page and ending with an ability to leverage existing services to provide new ones, that could fit better user requirements.

Objective of the work was to study the both of technologies in order to bring an implementation of the collaborative editor built with use of existing library for synchronization and manipulation with changes provided by the project Etherpad that will be integrated into JEE platform in a form of portlet application. The analysis brings many findings which sometimes resulted in choosing of another way of design. Many of these findings was caused by the lack of experiences in an area of both the technologies. The first task was to breakdown the infrastructure of the Etherpad Lite application. It was also the most crucial phase, however, it finally emerges the way, how to leverage the existing libraries to build upon them JEE application.

From the aspect of JEE application logic the theme impresses with many challenges from the possibilities of extension in a view of user experience, extension of Attributes to enable the inserting of images and finally, with very crucial topic – optimization.

However, due to lack of experience in the area of JavaScript server-side frameworks and many problems facing while trying to put together the part of the project with the JEE environment of JBoss AS, the implementation of the CE tool did not provide sufficient results. Most of the time costs the misleading analysis of origin objective to leverage more from the Etherpad Lite implementation and try to identify possible interface for attaching the JEE server side implementation.

The work on this paper brings me more than a wider overview of actually used technologies, experience with the development of project in the environment of JEE or importance of JavaScript role in todays web development, but mainly the knowledge that quality analysis needs to build wider knowledge background.

Bibliography

[1] ASHISH SARIN. *Portlets in Action*. 20 Baldwin Road PO Box 261 Shelter Island, NY 11964: Manning Publications Co., 2012. ISBN 9781935182542.

[2] RICHARDSON, W. Clay, Donald AVONDOLIO, Joe VITALE, Peter LEN a Kevin T. SMITH. *Professional portal development with open source tools: Java TM Portlet API, Lucene, James, Slide (Wrox Press)*. Indianapolis, Ind.: Wiley Pub., 2004, 400 p. ISBN 04-714-6951-3.

[3] ACE (editor). In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 2012-04-12 [cit. 2012-04-18]. from http://en.wikipedia.org/wiki/ACE_(editor)

[4] Collaborative real-time editor. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 2012-04-11 [cit. 2012-04-18]. from http://en.wikipedia.org/wiki/Collaborative_real-time_editor

[5] FOWLER, Martin. *Destilované UML*. 1st ed. Praha: Grada, 2009, Knihovna programátora (Grada). ISBN 978-80-247-2062-3.

[6] SARANG, Poornachandra. *Practical Liferay Java-based portal applications development*. Berkeley, Calif: Apress, 2009. ISBN 978-143-0218-487.

[7] KATZ, Max and Ilya SHAIKOVSKY. *Practical RichFaces*. 2nd ed. New York, N.Y.: Distributed by Springer Science Business Media, c2011, 392 p. Expert's voice in Java technology. ISBN 978-143-0234-500.

[8] COLLINS-SUSSMAN, Ben, Brian W. FITZPATRICK a C. Michael PILATO. *Version control with subversion*. 2nd ed. Beijing: O'Reilly, 2008, 404 p. ISBN 978-0-596-51033-6.

[9] JSR-168. *Java™ Portlet Specification*: *Version 1.0*. 2003. from: http://jcp.org/aboutJava/communityprocess/review/jsr168/index.html.

[10] JSR-286. *Java™ Portlet Specification*: *Version 2.0*. 2008. from: http://jcp.org/aboutJava/communityprocess/final/jsr286/index.html

Figures

Appendix A: Manual how to set up the project environment and start the application

To set up the environment from the added sources we need to forward this guideline:

1. Setup local mysql database server if not already present. This can be done in linux by executing following list of commands:

1. Sign in as root or use the fakeroot to be allowed to provide installation. Following commands works in Debian and Debian-based distributions, such as Ubuntu. In other environments there are also available packages with GUI installation wizards:

   linux@cmd:~$ sudo apt-get install mysql-server mysql-client

2. After successful installation log in to mysql database client with previously given root credentials:

   linux@cmd:~$ mysql -u root -p

   linux@cmd:~$ <type your root password>

   mysql> comand prompt should occurs.

3. Create new database by providing command:

   mysql> CREATE DATABASE portletepad;

4. Create new db user for our application; type:

   mysql> CREATE USER portletepad;

5. Create new db user for our application with all the privileges to created database:

   mysql> GRANT ALL ON portletepad.* TO 'portletepad'@'localhost' IDENTIFIED BY 'portletepad';

6. Copy the attached files into file system in to folder with read & write permissions permissions.

7. Now run the script placed in the main folder to build a database structure:

   mysql> source /path/to/script/database.sql

8. Now we should have set the database for the application to work. You can now logout from the

mysql console with one of the following commands:

mysql> quit or mysql> bye

2. We suppose that you have placed the attached files into your file system. The folder consists from GateIn Portal Container configured with the JBoss AS 6.0 and preconfigured Portal instance. Then the folder consists Java Development Kit (JDK) 1.7.0 to run the application server on it. Although, the GateIn Portal Container is configured to use the attached JDK, there is need to provide one configuration before the execution of the server. However, the JBoss needs to set up the JBOSS_HOME environment variable to point to the actual JBoss (with GateIn) instance to start properly. This can be done by executing the next dommands:

    1. export JBOSS_HOME=/path/to/GateInWithJBossAS

3. next we can start the server providing next steps:

    1. Walk into directory GateInWithJBossAS:

        linux@cmd:~$ cd /path/to/GateInWithJBossAS

    2. And run the server with next command:

        linux@cmd:~$ ./bin/run.sh

    3. The GateIn Portal Container build upon the JBoss AS should be after minute or two started – you will know this by the line ending with startup time information:

        ...JBossAS [6.0.0.Final "Neo"] Started in 42s:451ms

4. Now open the browser and navigate to the url: http://localhost:8080/portal

5. Sign in by clicking on the one predefined account.

6. Navigate from the top menu to the portlet: Home –> Portlet Etherpad Portal Page

7. The Portlet is opened and could be used The behaviour is intutive.

8. From the list of pads select one. The pad will open the editor with content of the pad.

9. Now you can edit th file and provide basic functionality od the editor.

10. For demonstration of collaborative behavior you must open the another browser and navigate to the portlet.

11. Sign in as another user.

12. Open the same pad as in the first instance and try if the remote changes are updating.

Appendix B: Definition of database structure and initial testing data in MySQL DDL

```
--

-- Table structure for table `Attribute`

--

DROP TABLE IF EXISTS `Attribute`;

CREATE TABLE `Attribute` (

  `id` int(10) NOT NULL AUTO_INCREMENT,

  `name` varchar(50) NOT NULL,

  PRIMARY KEY (`id`)

) ENGINE=MyISAM AUTO_INCREMENT=12 DEFAULT CHARSET=latin1;

LOCK TABLES `Attribute` WRITE;

INSERT INTO `Attribute` VALUES (1,'bold'),(2,'strikethrough'),(3,'italic'),
(4,'underline'),(5,'list'),(6,'list'),(7,'list'),(8,'list'),(9,'list'),
(10,'insertorder'),(11,'insertorder');

UNLOCK TABLES;

--

-- Table structure for table `AttributeItem`

--

DROP TABLE IF EXISTS `AttributeItem`;

CREATE TABLE `AttributeItem` (

  `id` int(10) NOT NULL AUTO_INCREMENT,

  `attribPoolId` int(10) NOT NULL,

  `number` int(3) NOT NULL,

  `attrId` int(10) NOT NULL,

  `value` varchar(50) NOT NULL,

  PRIMARY KEY (`id`),

  KEY `attribPoolId` (`attribPoolId`),
```

```
  KEY `attrId` (`attrId`)
) ENGINE=MyISAM AUTO_INCREMENT=3 DEFAULT CHARSET=latin1;
--
-- Dumping data for table `AttributeItem`
--
LOCK TABLES `AttributeItem` WRITE;
INSERT INTO `AttributeItem` VALUES (1,0,0,3,'true'),(2,2,0,1,'true');
UNLOCK TABLES;
--
-- Table structure for table `AttributePool`
--
DROP TABLE IF EXISTS `AttributePool`;
CREATE TABLE `AttributePool` (
  `id` int(10) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM AUTO_INCREMENT=3 DEFAULT CHARSET=latin1;
--
-- Dumping data for table `AttributePool`
--
LOCK TABLES `AttributePool` WRITE;
INSERT INTO `AttributePool` VALUES (1),(2);
UNLOCK TABLES;
--
-- Table structure for table `Changeset`
--
DROP TABLE IF EXISTS `Changeset`;
CREATE TABLE `Changeset` (
  `id` int(10) NOT NULL AUTO_INCREMENT,
  `number` int(10) NOT NULL,
```

```
  `rule` varchar(50) NOT NULL,
  `charbank` varchar(50) NOT NULL,
  `attribPoolId` int(10) NOT NULL,
  `padId` int(10) NOT NULL,
  `authorId` int(10) NOT NULL,
  `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  KEY `attribPoolId` (`attribPoolId`),
  KEY `padId` (`padId`),
  KEY `authorId` (`authorId`)
) ENGINE=MyISAM AUTO_INCREMENT=4 DEFAULT CHARSET=latin1;

--

-- Dumping data for table `Changeset`

--

LOCK TABLES `Changeset` WRITE;

INSERT INTO `Changeset` VALUES (1,1,'Z:0>7|*0+7','Welcome',0,1,0,'2012-05-01
20:41:16'),(2,2,'Z:7>1|=7*0|1+1','\n',0,1,0,'2012-05-01 20:41:16'),(3,3,'Z:8>3|
1=8*0+3','ano',1,1,1,'2012-05-01 20:41:16');

UNLOCK TABLES;

--

-- Table structure for table `Pad`

--

DROP TABLE IF EXISTS `Pad`;

CREATE TABLE `Pad` (
  `id` int(10) NOT NULL AUTO_INCREMENT,
  `name` varchar(50) NOT NULL,
  `creatorId` int(10) NOT NULL,
  `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name` (`name`),
```

```
  KEY `creatorId` (`creatorId`)
) ENGINE=MyISAM AUTO_INCREMENT=3 DEFAULT CHARSET=latin1;
--
-- Dumping data for table `Pad`
--
LOCK TABLES `Pad` WRITE;
INSERT INTO `Pad` VALUES (1,'karol pad',1,'2012-05-01 20:41:16'),(2,'alfonz
pad',2,'2012-05-01 20:41:16');
UNLOCK TABLES;
--
-- Table structure for table `Session`
--
DROP TABLE IF EXISTS `Session`;
CREATE TABLE `Session` (
  `id` int(10) NOT NULL AUTO_INCREMENT,
  `padId` int(10) NOT NULL,
  `userId` int(10) NOT NULL,
  `color` varchar(7) DEFAULT '#000000',
  `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `lastSeen` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
  PRIMARY KEY (`id`),
  UNIQUE KEY `padId` (`padId`,`color`),
  KEY `userId` (`userId`)
) ENGINE=MyISAM AUTO_INCREMENT=12 DEFAULT CHARSET=latin1;
--
-- Dumping data for table `Session`
--
LOCK TABLES `Session` WRITE;
INSERT INTO `Session` VALUES (1,1,1,'#123456','2012-05-19 09:33:41','2012-05-19
09:33:41'),(2,1,2,'#654321','2012-05-19 09:33:41','2012-05-19 09:33:41'),
```

```
(4,2,4,NULL,'2012-05-19   13:08:00','2012-05-20   09:20:21'),(5,1,4,NULL,'2012-05-19
13:10:55','2012-05-20    08:00:05'),(6,1,5,NULL,'2012-05-19    19:17:17','2012-05-20
08:00:15'),(7,2,6,NULL,'2012-05-20         03:03:57','2012-05-20         03:51:47'),
(8,1,6,NULL,'2012-05-20   03:25:30','2012-05-20   06:00:34'),(9,1,7,NULL,'2012-05-20
06:15:39','2012-05-20   07:59:10'),(10,2,7,NULL,'2012-05-20   06:38:18','2012-05-20
07:22:21'),(11,2,5,NULL,'2012-05-20 07:58:34','2012-05-20 07:59:30');

UNLOCK TABLES;

--

-- Table structure for table `User`

--

DROP TABLE IF EXISTS `User`;

CREATE TABLE `User` (

  `id` int(10) NOT NULL AUTO_INCREMENT,

  `name` varchar(50) NOT NULL,

  `externId` varchar(50) DEFAULT NULL,

  `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,

  PRIMARY KEY (`id`)

) ENGINE=MyISAM AUTO_INCREMENT=8 DEFAULT CHARSET=latin1;

--

-- Dumping data for table `User`

--

LOCK TABLES `User` WRITE;

INSERT     INTO     `User`     VALUES     (1,'jonas',NULL,'2012-05-19     09:34:03'),
(2,'karol',NULL,'2012-05-19   09:34:03'),(3,'alfonz',NULL,'2012-05-19   09:34:03'),
(4,'root',NULL,'2012-05-19     11:26:45'),(5,'john',NULL,'2012-05-19     19:17:15'),
(6,'demo',NULL,'2012-05-20 03:03:39'),(7,'mary',NULL,'2012-05-20 06:15:34');

UNLOCK TABLES;
```

Appendix C: Object-relational mapping configuration with Hibernate mapping files

Mapping document version

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
```

Pad.hbm.xml

```
<hibernate-mapping>
  <class name="org.webepad.model.Pad" table="Pad" lazy="false">
    <id name="id" type="java.lang.Long">
      <column name="id" />
        <generator class="identity" />
    </id>
    <property name="name" type="java.lang.String">
      <column name="name" />
    </property>
    <property name="created" type="java.util.Date">
      <column name="created" />
    </property>
    <many-to-one name="creator" class="org.webepad.model.User"
      fetch="join">
      <column name="creatorId" />
    </many-to-one>
    <bag name="changesets" inverse="false" table="Changeset" lazy="true"
      order-by="number asc">
      <key column="number" />
      <one-to-many class="org.webepad.model.Changeset" />
    </bag>
    <map name="userSessions" inverse="false" table="Session" lazy="true"
      access="field">
      <key column="padId" />
        <map-key column="userId" type="java.lang.Long"></map-key>
          <one-to-many class="org.webepad.model.Session" />
```

```
        </map>
  </class>
</hibernate-mapping>
```

User.hbm.xml

```
<hibernate-mapping>
  <class name="org.webepad.model.User" table="User" lazy="false">
    <id name="id" type="java.lang.Long">
      <column name="id" />
      <generator class="identity" />
    </id>
    <property name="externId" type="java.lang.String" insert="false"
      update="false">
      <column name="externId" />
  </property>
    <property name="name" type="java.lang.String">
      <column name="name" />
    </property>
    <map name="padSessions" inverse="false" table="Session" lazy="true"
      access="field">
      <key>
        <column name="padId" />
      </key>
      <map-key type="java.lang.Long"></map-key>
      <one-to-many class="org.webepad.model.Session" />
    </map>
  </class>
</hibernate-mapping>
```

Changeset.hbm.xml

```
<hibernate-mapping>
  <class name="org.webepad.model.Changeset" table="Changeset" lazy="false">
    <id name="id" type="java.lang.Long">
      <column name="id" />
      <generator class="identity" />
    </id>
    <property name="rule" type="java.lang.String">
      <column name="rule" />
```

```
    </property>
    <property name="charbank" type="java.lang.String">
      <column name="charbank" />
    </property>
    <property name="number" type="java.lang.Integer">
      <column name="number" />
    </property>
    <property name="created" type="java.util.Date">
      <column name="created" />
    </property>
    <many-to-one name="pad" class="org.webepad.model.Pad" fetch="join">
      <column name="padId" />
    </many-to-one>
    <many-to-one name="author" class="org.webepad.model.User"
             fetch="join">
      <column name="authorId" />
    </many-to-one>
    <one-to-one
      name="attributePool" class="org.webepad.model.AttributePool">
    </one-to-one>
  </class>
</hibernate-mapping>
```

Session.hbm.xml

```
<hibernate-mapping>
  <class name="org.webepad.model.Session" table="Session" lazy="false">
    <id name="id" type="java.lang.Long">
      <column name="id" />
      <generator class="identity" />
    </id>
    <property name="colorCode" type="java.lang.String">
      <column name="color" />
    </property>
    <property name="created" type="java.util.Date">
      <column name="created" />
    </property>
    <property name="lastSeen" type="java.util.Date">
```

```
      <column name="lastSeen" />
    </property>
    <many-to-one name="pad" class="org.webepad.model.Pad" fetch="join">
      <column name="padId" />
    </many-to-one>
    <many-to-one name="user" class="org.webepad.model.User" fetch="join">
      <column name="userId" />
    </many-to-one>
  </class>
</hibernate-mapping>
```

AttributePool.hbm.xml

```
<hibernate-mapping>
  <class name="org.webepad.model.AttributePool" table="AttributePool"
lazy="false">
    <id name="id" type="java.lang.Long">
      <column name="id" />
      <generator class="identity" />
    </id>
    <one-to-one name="changeset" class="org.webepad.model.Changeset"></one-
to-one>
    <map name="attributeMap" table="AttributeItem" lazy="true"
access="field">
      <key>
        <column name="number" />
      </key>
      <map-key type="java.lang.Integer"></map-key>
      <one-to-many class="org.webepad.model.AttributeItem" />
    </map>
  </class>
</hibernate-mapping>
```

AttributeItem.hbm.xml

```
<hibernate-mapping>
  <class name="org.webepad.model.AttributeItem" table="AttributeItem"
      lazy="false">
    <id name="id" type="java.lang.Long">
      <column name="id" />
```

```
      <generator class="identity" />
    </id>
    <property name="number" type="java.lang.Integer">
      <column name="number" />
    </property>
    <many-to-one name="attribute" class="org.webepad.model.Attribute"
fetch="join">
      <column name="attrId" />
    </many-to-one>
    <property name="value" type="java.lang.String">
      <column name="value" />
    </property>
    <many-to-one name="attributePool"
      class="org.webepad.model.AttributePool" fetch="join">
      <column name="attribPoolId" />
    </many-to-one>
  </class>
</hibernate-mapping>
```

Attribute.hbm.xml

```
<hibernate-mapping>
  <class name="org.webepad.model.Attribute" table="Attribute" lazy="false">
    <id name="id" type="java.lang.Long">
      <column name="id" />
      <generator class="identity" />
    </id>
    <property name="name" type="java.lang.String">
      <column name="name" />
    </property>
  </class>
</hibernate-mapping>
```