

DECA

Document editing, commenting and analysis system

Technical specification

25.03.2002

**Antti Luomi
Markus Rautopuro
Matti Öhman**

This page has been intentionally left blank.

TABLE OF CONTENTS

1.	Introduction	5
1.1	Version history.....	5
1.2	The purpose of this document	5
1.3	The product and the environment.....	5
1.4	Definitions, abbreviations and notations	6
1.5	References	6
2.	Overview of the system.....	7
2.1	Software type definition.....	7
2.2	Environment	7
2.3	Implementational constraints	7
2.4	Complied standards	7
3.	System architecture	8
3.1	Design principles	8
3.2	The information store	8
3.3	Architecture of the software	9
3.3.1	Overview	9
3.3.2	Data path from database to users	9
3.3.2.1	Information pipeline illustration	9
3.3.2.2	Database layer	11
3.3.2.3	Factory layer.....	11
3.3.2.3.1	General functionality	11
3.3.2.3.2	Document factory.....	11
3.3.2.3.3	Document list factory	11
3.3.2.3.4	Document version list factory.....	12
3.3.2.3.5	User settings factory.....	12
3.3.2.4	Filter layer.....	12
3.3.2.4.1	Version filter	12
3.3.2.4.2	Privilege filter.....	13
3.3.2.4.3	Already read filter	13
3.3.2.5	View layer	13
3.3.2.5.1	General functionality	13
3.3.2.5.2	Document view.....	14
3.3.2.5.3	Content view	14
3.3.2.5.4	Comment view	14
3.3.2.5.5	Document list view	14
3.3.2.5.6	Version selector view	14
3.3.3	Data path from users to database.....	14
3.3.4	Version control	14
3.3.5	Access control.....	14
3.3.6	User dependent views	15
4.	Class definitions	17
4.1	Packages.....	17
4.1.1	Overview	17
4.1.2	Package struct.....	18
4.1.3	Package version.....	18
4.1.4	Package view and package user.....	19
4.1.5	Package access.....	20
4.1.6	Package general	21
4.2	Information entities	21
4.2.1	Overview	21
4.2.2	Documents	21

Luomi, Rautopuro, Öhman

25.03.2002

4.2.3	Templates	23
4.2.4	Version and access control.....	24
4.2.4.1	Version control	25
4.2.4.2	Access control.....	25
5.	Error handling.....	26
5.1	Overview.....	26
5.2	Logging.....	26
6.	Discarded approaches.....	27
6.1	Relational database as storage format.....	27
6.1.1	Reasons for discarding	27
6.1.2	Versioning principles	27
6.1.3	Representing trees in SQL tables.....	27
6.1.4	Database tables	28
6.1.4.1	Table document.....	28
6.1.4.2	Table structure	28
6.1.4.3	Table content.....	29
6.1.4.4	Table comment.....	29
6.1.4.5	Table user [under construction]	30
6.2	XML as storage format for documents.....	30
7.	Open issues	31
7.1	User interface issues.....	31
7.2	Missing class descriptions	31

Luomi, Rautopuro, Öhman

25.03.2002

1. INTRODUCTION

1.1 Version history

Version	Date	Author	Description
0.1	15.11.2001	MRa & ALu	Created.
0.3	02.12.2001	MRa & ALu	Table + class descriptions.
0.5	09.12.2001	MRa & ALu	MÖh's + TKä's comments.
0.7	10.12.2001	MRa & ALu	Added introduction and overview.
1.0	11.12.2002	MRa & ALu	Released for public inspection.
1.1	11.02.2002	MRa	Made some additions to error handling section + multi user section
1.2	25.03.2002	MRa	Phase 3 implementational additions

1.2 The purpose of this document

This document covers the initial technical details of the implementation of the DECA project. Particularly the system architecture, information storage solution and Java class hierarchy are presented here.

1.3 The product and the environment

The product and the environment are described in [reqspec] (see section 1.5).

The working title of the software is Document Editing, Commenting and Analysis system or DECA for short.

The goal is to produce a system that can be used by various organizations and communities to increase the accessibility of documents and to make it easier to give and receive feedback related to these documents. The software is designed and implemented especially for Teamware's needs. The feedback from Teamware is needed to make the software to match real-world needs i.e. as useful as possible.

The goals of the software are fourfold:

1. documenting information
2. commenting documents
3. organizing comments
4. continuous development of documents within an organization.

The aim is to produce a comprehensive management system for document development process. It strives to offer centralized solution, which is easy to use and generic enough so that it can meet different and changing needs. In addition the creation and managed development process of structured documents is supported to template and version management features.

Teamware will use the software for managing "segment description" and related "case description" documents. The system will help the communication between marketing, sales and product development departments. The software will make it easier to send feedback and simplify the management of the received feedback. By comparing the documents and the obtained feedback it is possible to analyze the deficiencies of the documents and to develop them further.

Luomi, Rautopuro, Öhman

25.03.2002

To fulfill these requirements the software is implemented as network application, which can be used with the most popular browsers. The implementation technologies will be selected so that the software will be as portable as possible.

1.4 Definitions, abbreviations and notations

These are defined in a separate document, see Vaatimusmäärittely, 1.3 Määritelmät, termit ja lyhenteet.

1.5 References

[4b4o] db4o – database for objects (english)

<http://www.db4o.com>

[trees] Trees in SQL (english)

http://searchdatabase.techtarget.com/tip/1,289483,sid13_gci537290,00.html

[pl@za] Teamware Pl@za 3.4 Fact Sheet (english)

<http://www.teamware.com/plaza/fs-plaza.htm>

[reqspec] DECA vaatimusmäärittely (finnish)

<http://www.hut.fi/~tamatti2/ohjtyo/Vaatimusmaarittely/>

[funcspec] DECA functional specification (english)

<http://www.hut.fi/~tamatti2/ohjtyo/FunctionalSpecification/>

[DecaJavaDoc] DECA JavaDoc documentation (english)

<http://www.hut.fi/~tamatti2/ohjtyo/javadoc.pdf>

Luomi, Rautopuro, Öhman

25.03.2002

2. OVERVIEW OF THE SYSTEM

2.1 Software type definition

DECA is a multitier enterprise application based on J2EE standard defined by Sun Microsystems, Inc.

2.2 Environment

The software will run on a web-server that has Tomcat 4.0, Sun JDK 1.3.1 and db4o 1.5 properly installed. The generated pages will be tested on the most popular web browsers. The development environment will also include Jakarta ANT 1.4.1.

2.3 Implementational constraints

Display size on some portables is limited to 800x600 pixels and the software should run even on these machines.

The browser requirements are: Microsoft Internet Explorer 5.0 or newer, or Netscape Navigator 4.5 or newer.

2.4 Complied standards

TCP (RFC 793)

<http://www.ibiblio.org/pub/docs/rfc/rfc793.txt>

HTTP 1.0 (RFC 1945)

<http://www.ietf.org/rfc/rfc1945.txt>

HTML 4.01 Specification, W3C Recommendation 24 December 1999

3. SYSTEM ARCHITECTURE

3.1 Design principles

The general structure of the system architecture follows J2EE standard defined by Sun Microsystems, Inc. The architecture is illustrated in figure 3-1.

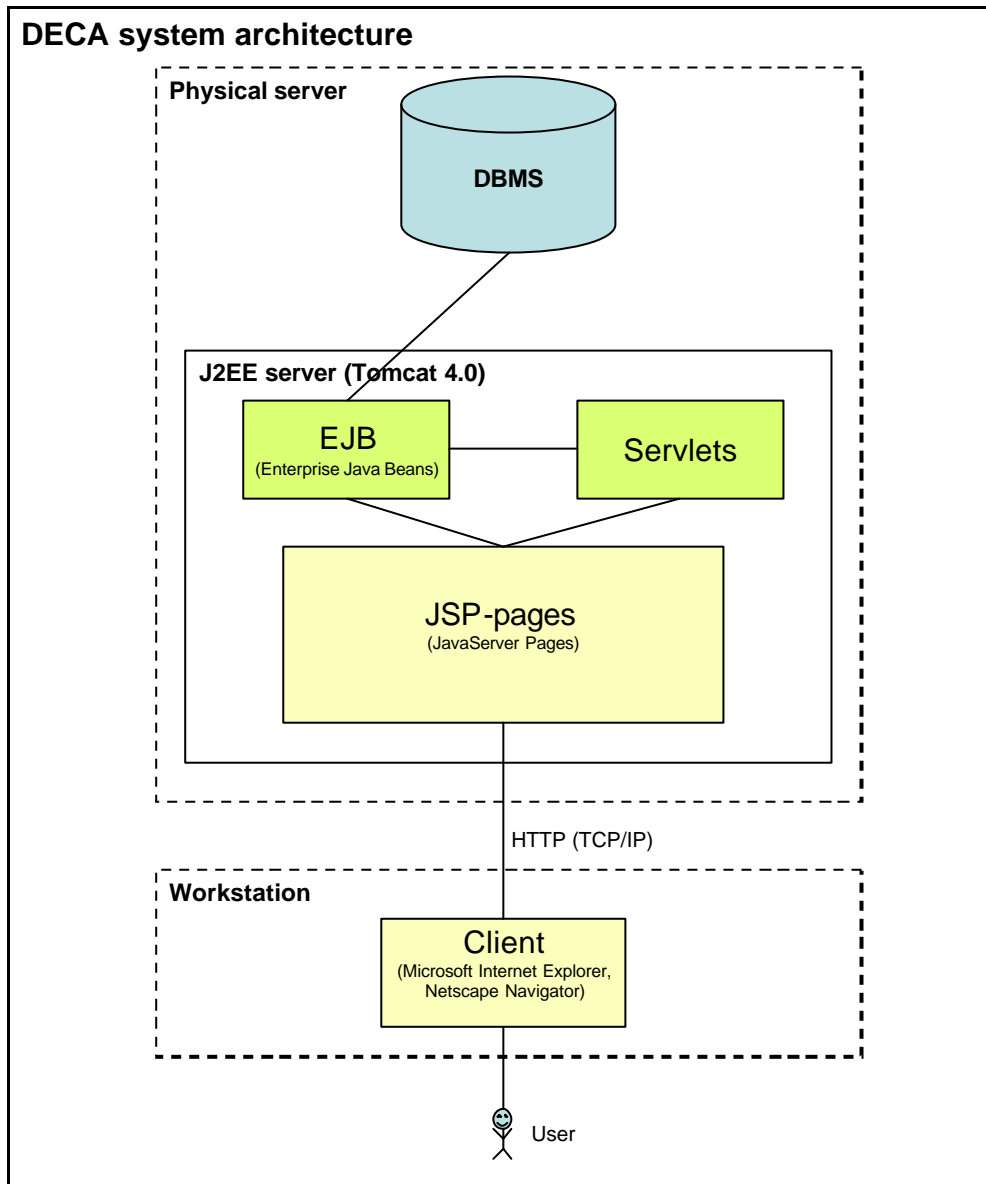


Figure 3-1. DECA system architecture

J2EE defines an ideal architecture for web-based multi-user applications. Jakarta Tomcat was chosen because it is also free and follows J2EE strictly.

3.2 The information store

The DBMS used in the DECA system is db4o, which is an object database engine. db4o offers an easy interface to submit database operations in Java. Figure 3-1 illustrates the difference between relational and object query implemented in Java.

Db4o	SQL
<pre>db4o.set(contract);</pre>	<pre>String sql = ""; int personID = generateID(statement); sql = "INSERT INTO brm_PERSON (ID, NAME) VALUES (" + Integer.toString(personID) + ", '" + contract.customer.name + "'" + ")"; statement.execute(sql); int customerID = generateID(statement); sql = "INSERT INTO brm_CUSTOMER (ID, PERSONID, " + "CUSTOMERNO) VALUES (" + Integer.toString(customerID) + ", " + Integer.toString(personID) + ", '" + contract.customer.customerNo + "'" + ")"; statement.execute(sql); int documentID = generateID(statement); sql = "INSERT INTO brm_DOCUMENT (ID, TITLE, " + "CREATIONDATE) VALUES (" + Integer.toString(documentID) + ", '" + contract.title + "'" + ", '" + Long.toString(contract.creationDate.getTime()) + "'"; statement.execute(sql); int contractID = generateID(statement); sql = "INSERT INTO brm_CONTRACT (ID, DOCUMENTID, " + "CONTRACTNO, SUBJECT, CUSTOMERID) VALUES (" + Integer.toString(contractID) + ", " + Integer.toString(documentID) + ", '" + contract.contractNo + "'" + ", '" + contract.subject + "'" + ", " + Integer.toString(customerID) + ")"; statement.execute(sql);</pre>

Figure 3-2. Code excerpt from function “save object”

In object databases there is no difference between database design and object design; objects are stored as is to the database. Therefore the system architecture design must be focused on designing appropriate object hierarchy.

3.3 Architecture of the software

3.3.1 Overview

The software architecture is based on object-oriented Java technology created by Sun Microsystems, Inc.

Software architecture is divided in three major sets of Java classes: factory classes, filter classes and view classes. In addition, several utility classes may be needed.

3.3.2 Data path from database to users

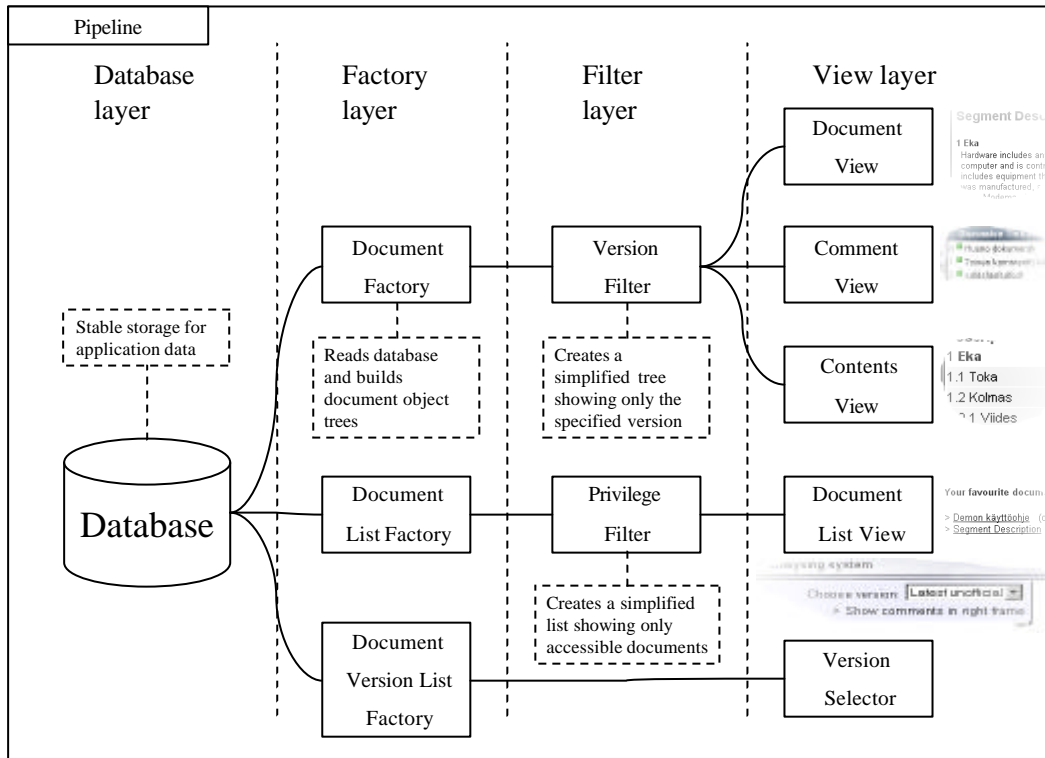
3.3.2.1 Information pipeline illustration

The data is processed in several layers before it is represented to users. This is illustrated in figure 3-3. The purpose of layered structure is to split the process in simple entities. This

Luomi, Rautopuro, Öhman

25.03.2002

helps multiple developers to implement functionality simultaneously and provides easier basis for testing.



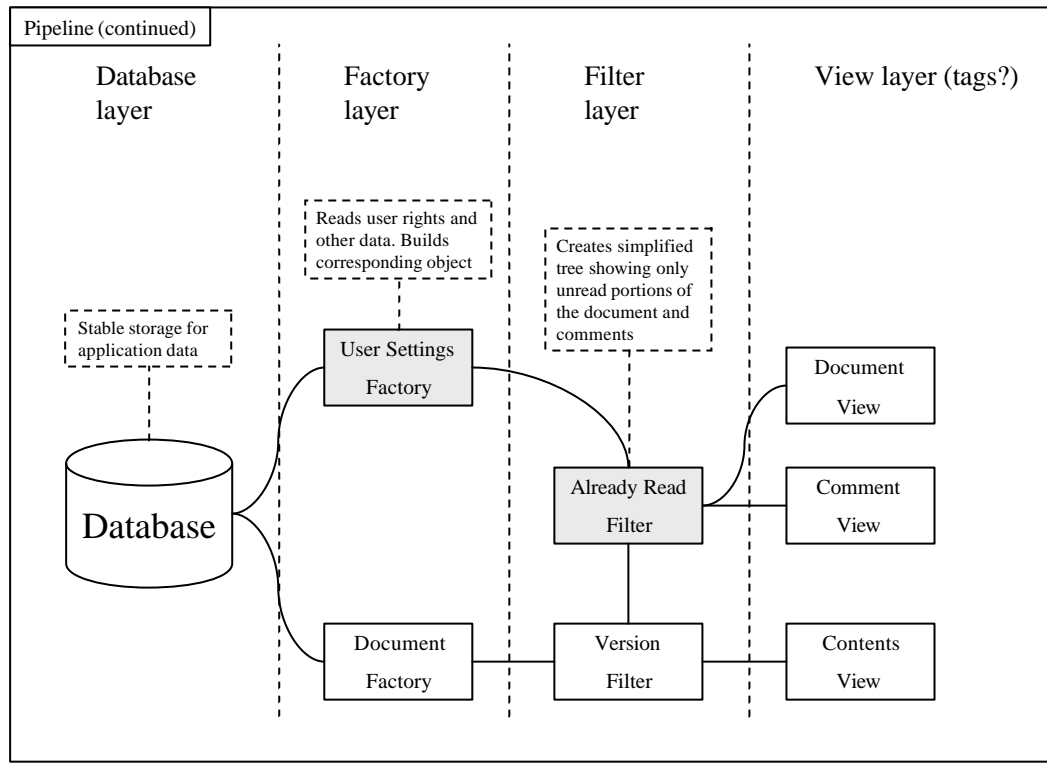


Figure 3-3. Data flow in the DECA system

3.3.2.2 Database layer

The physical database engine, db4o. All the data of the system is permanently stored here.

3.3.2.3 Factory layer

3.3.2.3.1 General functionality

The function of factory layer is to make queries to database and create instances of Java classes from the fetched data. The layer contains a factory for each type of query. These factories are described below.

3.3.2.3.2 Document factory

Document factory queries the database for certain document. It creates an object that contains the document metadata, all versions of it's contents and all the comments related to contents. This query is used when the user wants to read a document.

3.3.2.3.3 Document list factory

Document list factory queries the database for metadata of all the documents the system contains and creates a list object which contains them all. This query is used when the user wants to get a list of the documents she can read.

Luomi, Rautopuro, Öhman

25.03.2002

3.3.2.3.4 Document version list factory

Document version list factory queries the database for all the versions of certain document the system contains and creates a list object which contains the version IDs. This query is used when the user wants to get a list of the versions a single document has.

3.3.2.3.5 User settings factory

User settings factory queries the database for info of particular user and creates corresponding object. This query is used by the system when it checks if the user has the right to perform a certain action.

3.3.2.4 Filter layer

Because the queries made by factory layer return lots of data (all the versions of one document, all document metadata in system, etc.), we need a mechanism to choose the right information from it to display to user. This is the main purpose of the filter layer. The layer contains several filters of different type which are described below. Notice that use of filters can be nested, so that output of one filter can be used as a input for another filter.

3.3.2.4.1 Version filter

The version filter is used to refine the data returned by the document factory (see 3.3.2.2.1). It processes the document object from document factory (which contains all the versions of the document) and returns the object which is reduced to represent a specific version of the document.

Figure 3-4 illustrates the functionality of the version filter.

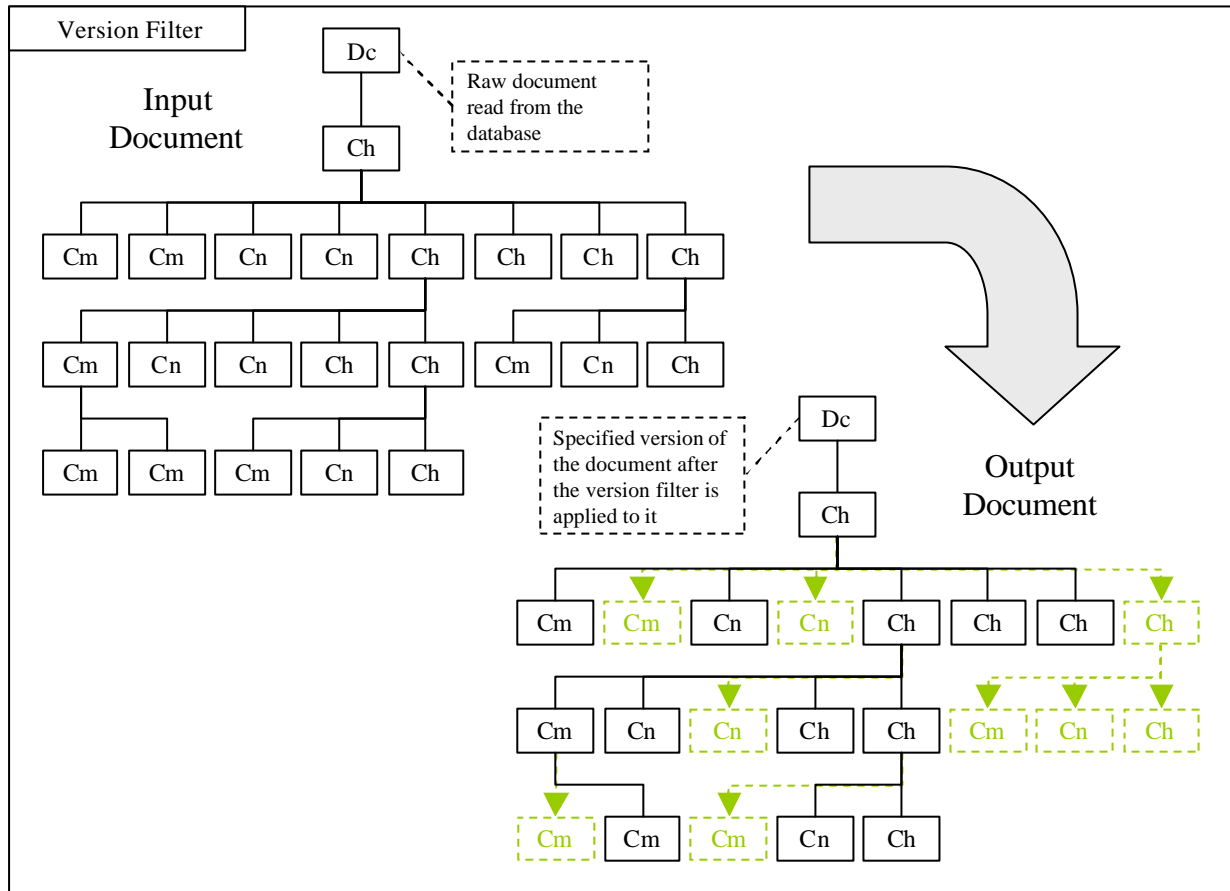


Figure 3-4. Version filtering principle

3.3.2.4.2 Privilege filter

The privilege filter is user to refine the data returned by the document list factory. It processes the document list object from the document list factory (which contains metadata for all the documents in the system) and returns the object which is reduced to a list of documents which are accessible for particular user.

3.3.2.4.3 Already read filter

The already read filter is used to filter data from document and document list factories with the information from user settings factory. It removes those parts of the input data that have already been read by the requesting user. Usually already read filter is used to filter data that already has been refined by another filter.

3.3.2.5 View layer

3.3.2.5.1 General functionality

The view layer contains mechanisms to represent filtered data to the user. View layer components are used in constructing the user interface. The following specification

Luomi, Rautopuro, Öhman

25.03.2002

concentrates on describing the types of data the UI components represent and how they interact with the user. The actual look and feel of the user interface is not to be described.

3.3.2.5.2 Document view

Displays the contents and chapter titles of the specified version of the whole document.

3.3.2.5.3 Content view

Shows the structure of the document version user is currently reading or modifying. This view is used to navigate in the document view.

3.3.2.5.4 Comment view

Displays the comment titles and comment contents of the document that user is currently working with.

3.3.2.5.5 Document list view

Document list view shows a list of the documents available in the DECA system using constraints specified by the user.

3.3.2.5.6 Version selector view

Version selector view shows a list of all the versions IDs of a single document.

3.3.3 Data path from users to database

The object oriented database engine makes updating data very simple. When the user modifies the data, all the changes are made to objects that can be stored to the database as is.

In multiuser environment, it is possible that two or more users are simultaneously modifying the same data object. To prevent problems caused by this scenario, a mechanism that allows only one modifier per data object will be implemented.

3.3.4 Version control

Version control in the DECA system is based on labeling different revisionable elements with a common version label. Revisionable elements in the DECA system are chapter, comment and content objects. Version labels are assigned to documents and templates. A version label is actually a timestamp.

Revisioning is implemented by giving *in* and *out* timestamps to each versionable element. *In* timestamp defines the moment when the element becomes valid. Correspondingly, *out* timestamp defines the invalidation time of the element. This approach makes it possible to fetch documents (including comments) as they were in any given time.

3.3.5 Access control

The DECA system supports access management. Documents (and templates) can be exploited in multiple ways. Some users can modify the documents, some can comment them and some can analyze them. Access to certain documents can even be totally denied for some users.

Access control solution in the DECA system is based on user groups. Users can belong to any number of groups. Documents (or templates) itself don't contain any access lists. Access control is described later on this document. Figure 3-5 shows the access control from the user's point of view.

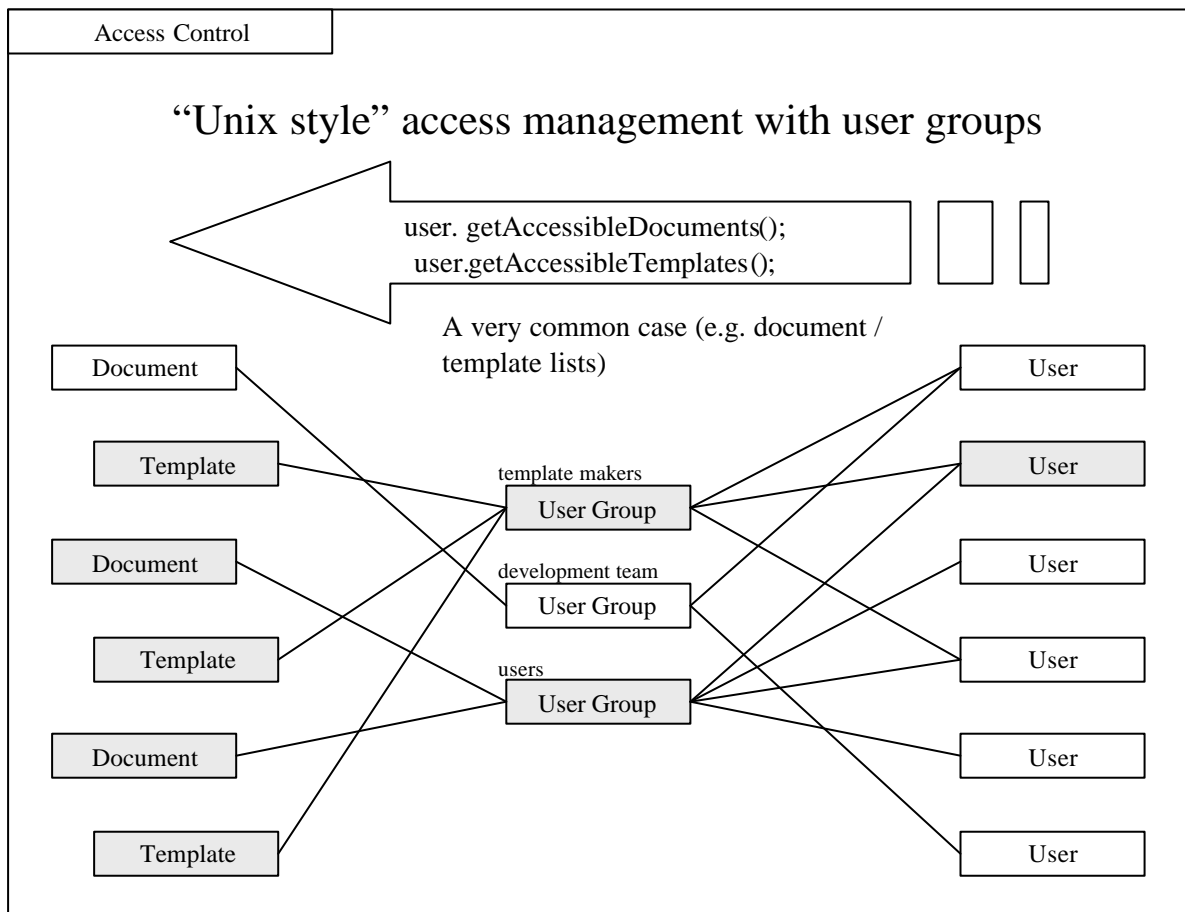


Figure 3-5. Access control from users view

3.3.6 User dependent views

The DECA system is designed to remember which chapters/comments the user has read. Figure 3-6 illustrates the idea of storing this information. Every row in the matrix represents a comment or a version of a chapter. Every column represents the users. When a user reads a comment or a chapter, the corresponding intersection point is checked in the matrix.

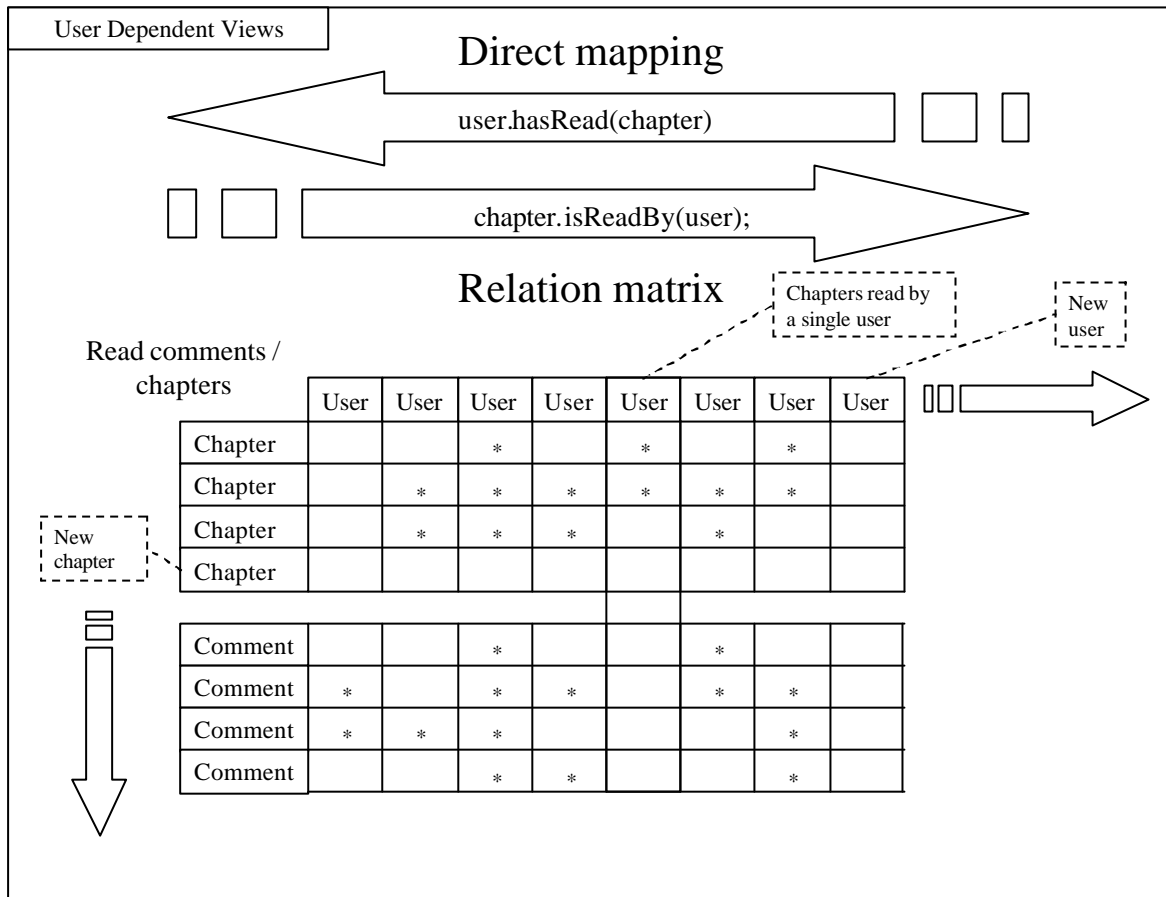


Figure 3-6. Usage behavior stored in a matrix

However, a matrix like this can grow very rapidly and data retrieval can be inefficient. Rather than storing the information in a complete matrix, it will be stored in user-specific parts (columns). Figure 3-7 shows the details on how the data is stored.

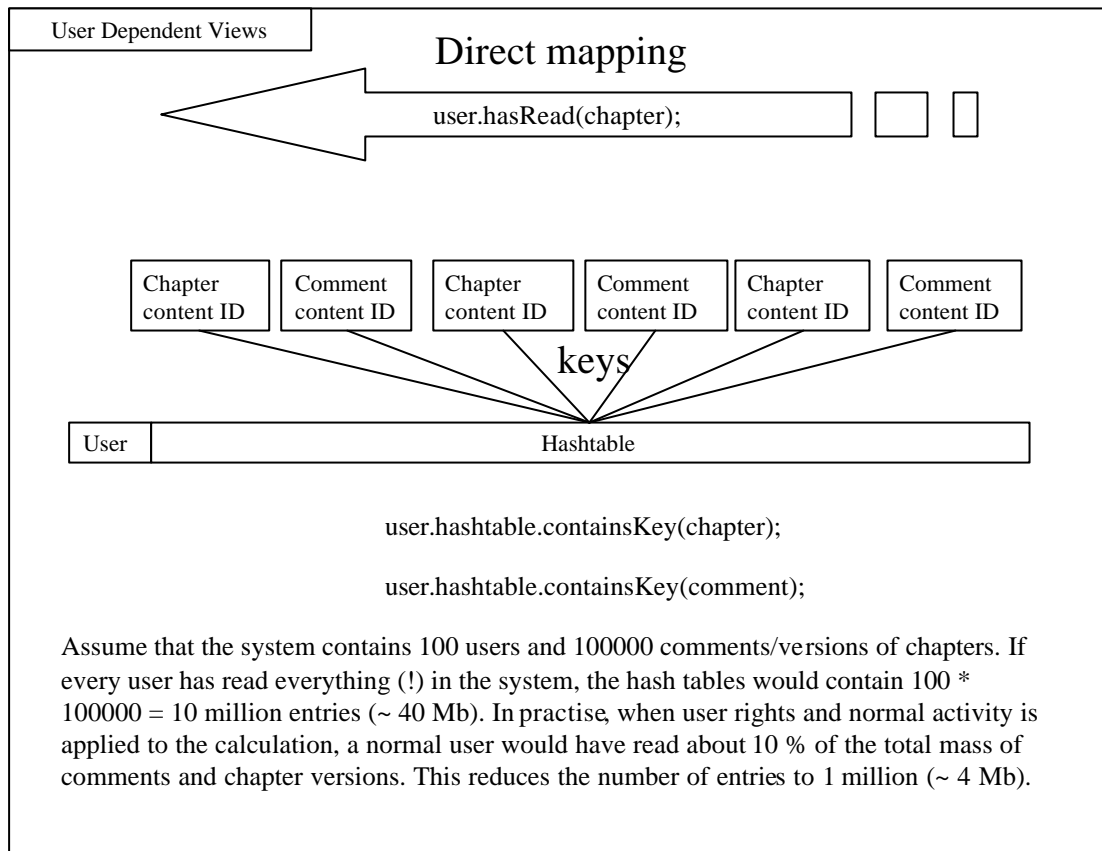


Figure 3-7. Usage behavior in practise

Since hash tables are used as information storage, only information about “if user has read a chapter/comment” needs to be stored. If hasn’t read some chapter/comment, no information will be present in the hash table.

This approach has several advantages. Adding new users becomes easy, since only an empty hash table will be assigned to a user. On the other hand, adding new comments/chapters make no modification to existing hash tables, since no one can possibly have read them.

3.3.7 Multiuser support

The DECA system will support multiple concurrent users. The number of simultaneous sessions is not limited by the software itself. Multiuser support is described further in the user interface specification.

4. CLASS DEFINITIONS

4.1 Packages

4.1.1 Overview

Chapter 4 describes only class relations in packages and will not go into implementation details.

There is no need in this document to specify the class structures of the filter elements, since they do not contain any special structure. All the filters can even be implemented in single class in different static methods.

A more detailed class description can be found from [DecaJavaDoc].

4.1.2 Package struct

Package struct contains all the classes that are needed to represent the basic elements of the DECA system. These elements are documents, templates, chapters and comments. Figure 4-1 contains the inheritance diagram of the classes in package struct.

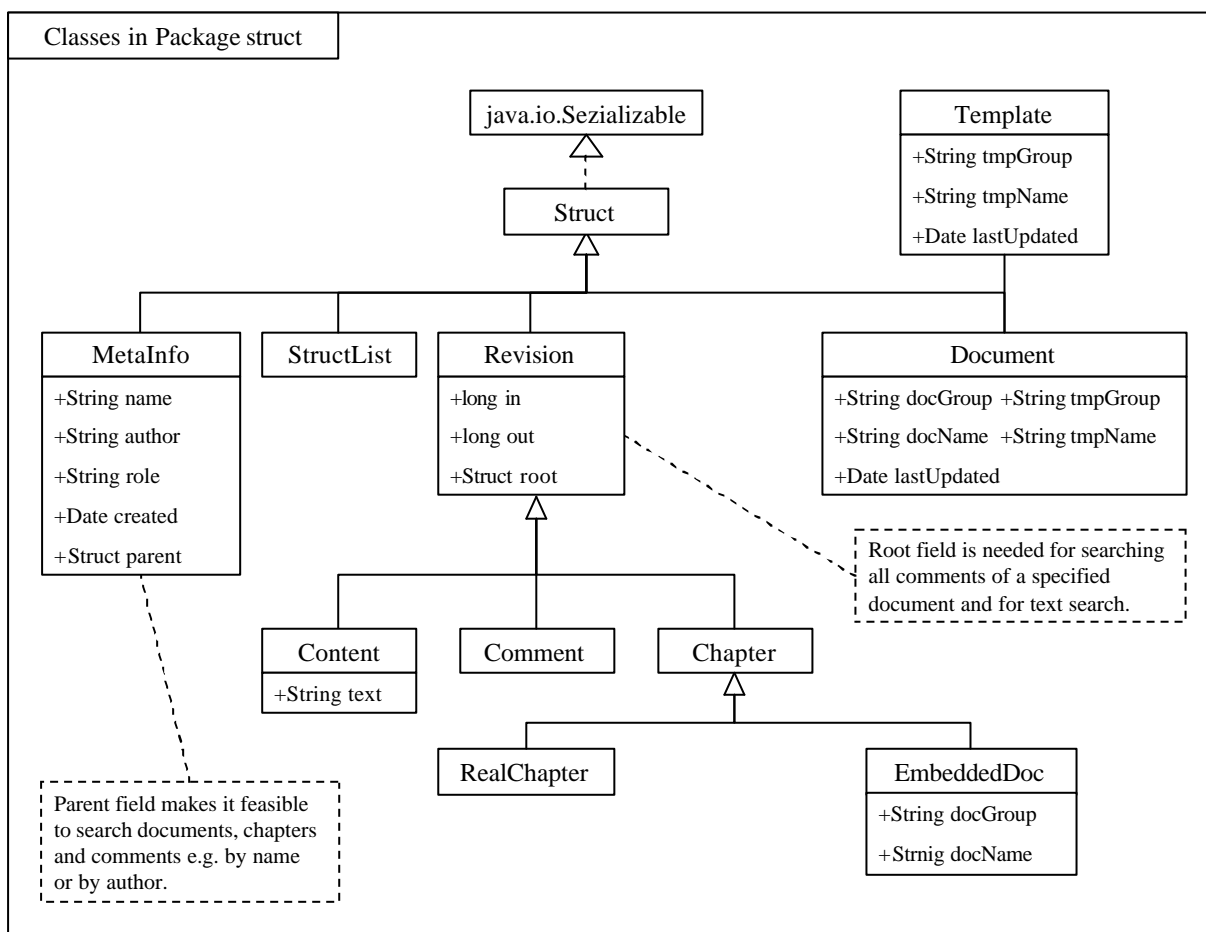


Figure 4-1. Package struct class inheritance diagram

4.1.3 Package version

Package version contains all the classes necessary to implement version labeling of documents and templates. Figure 4-2 contains the inheritance diagram of the classes in package version.

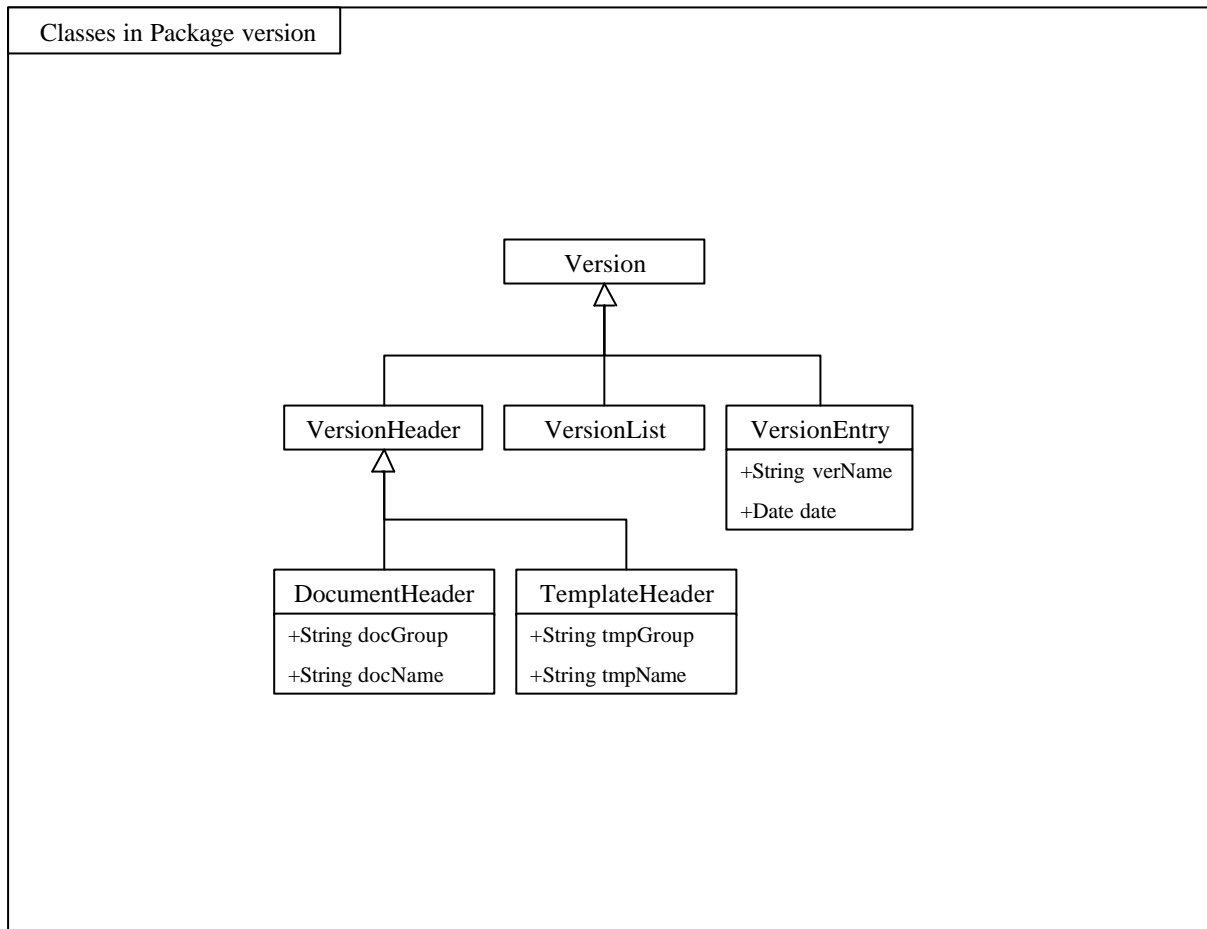


Figure 4-2. Package version class inheritance diagram

4.1.4 Package view and package user

Package view contains classes for user-dependent view generation. This functionality is described in section 3.2.2. Package user contains classes for storing metadata for users (passwords, etc.). Figure 4-3 contains the inheritance diagram of the classes in both packages.

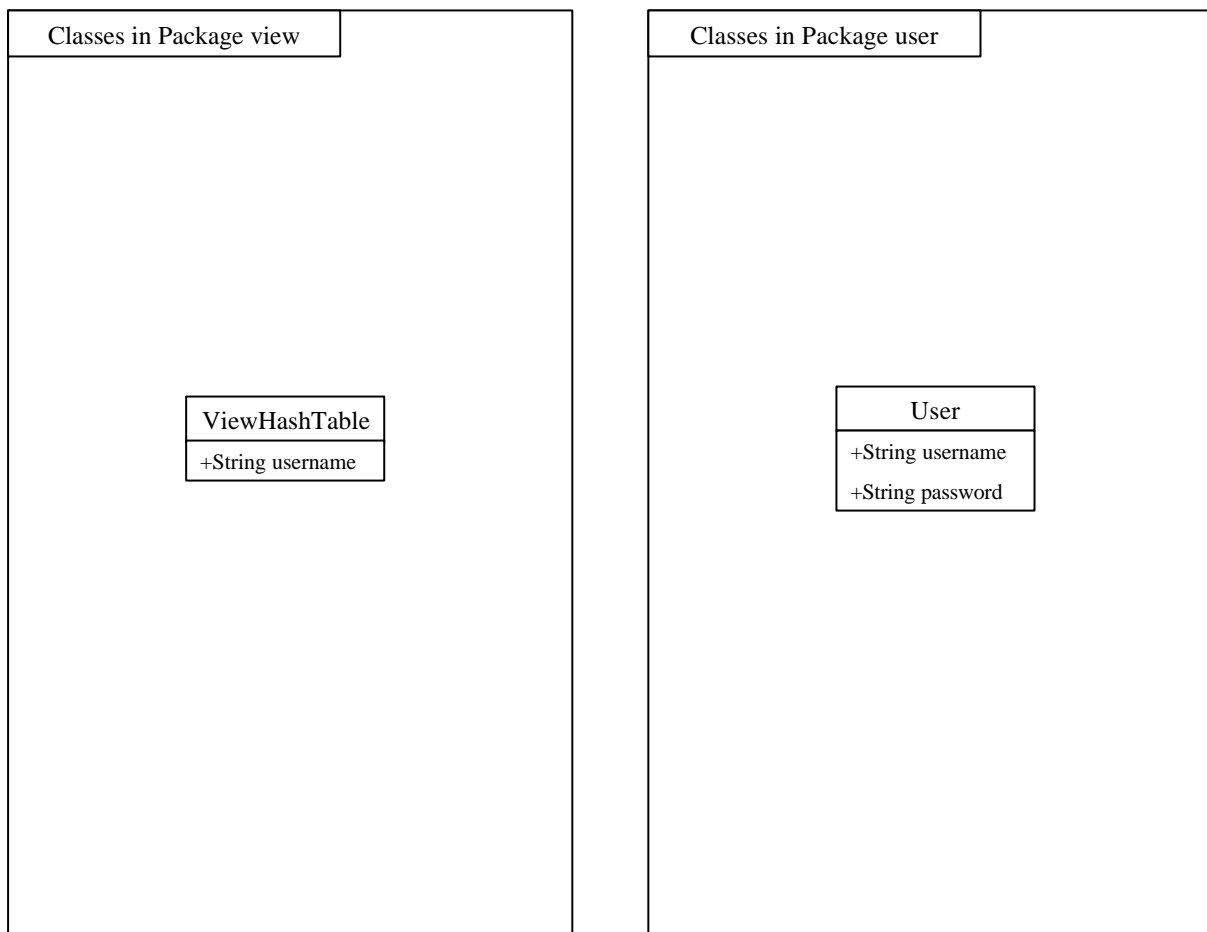


Figure 4-3. Class inheritance diagram for packages view and user

4.1.5 Package access

Package version encapsulates the functionality of access control in the DECA system. Access control is described in section 4.2.4. Figure 4-4 contains the inheritance diagram of the classes in package access.

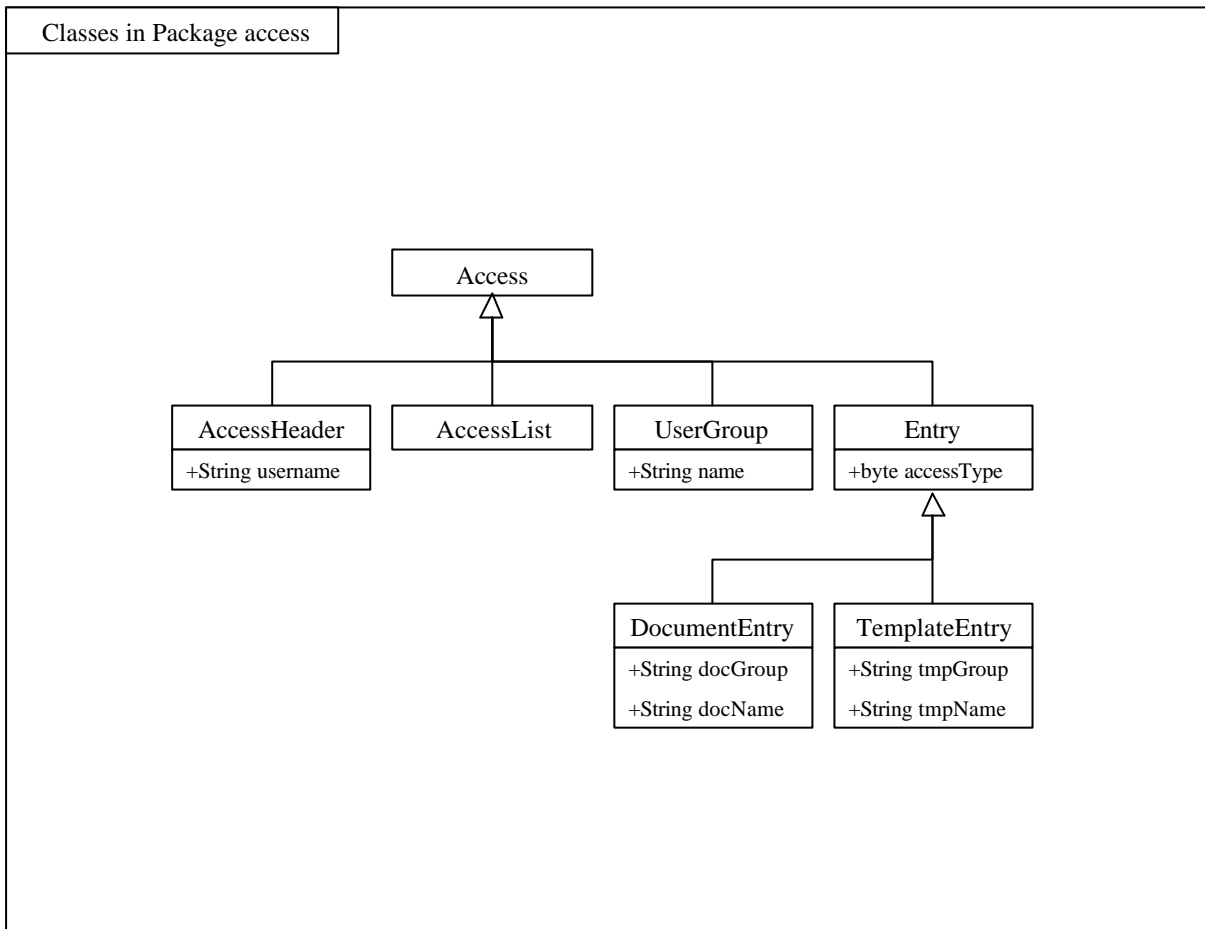


Figure 4-4. Class inheritance diagram for package access

4.1.6 Package general

Package general will be used to hold the any common classes that are used in the system.

4.2 Information entities

4.2.1 Overview

Section 4.2 describes the actual object entities that reside in the object database. These object entities (for example documents, templates) are the data itself.

4.2.2 Documents

The structure of a document in the DECA system is shown in figure 4-5. A document contains meta data about the document itself (author, document name, document group, date created etc.). The document contains one root chapter. The root chapter includes a list of contents (all the versions of this chapter's content), a list of comments and a list of sub-chapters. Comments include one content and a list of sub-comments.

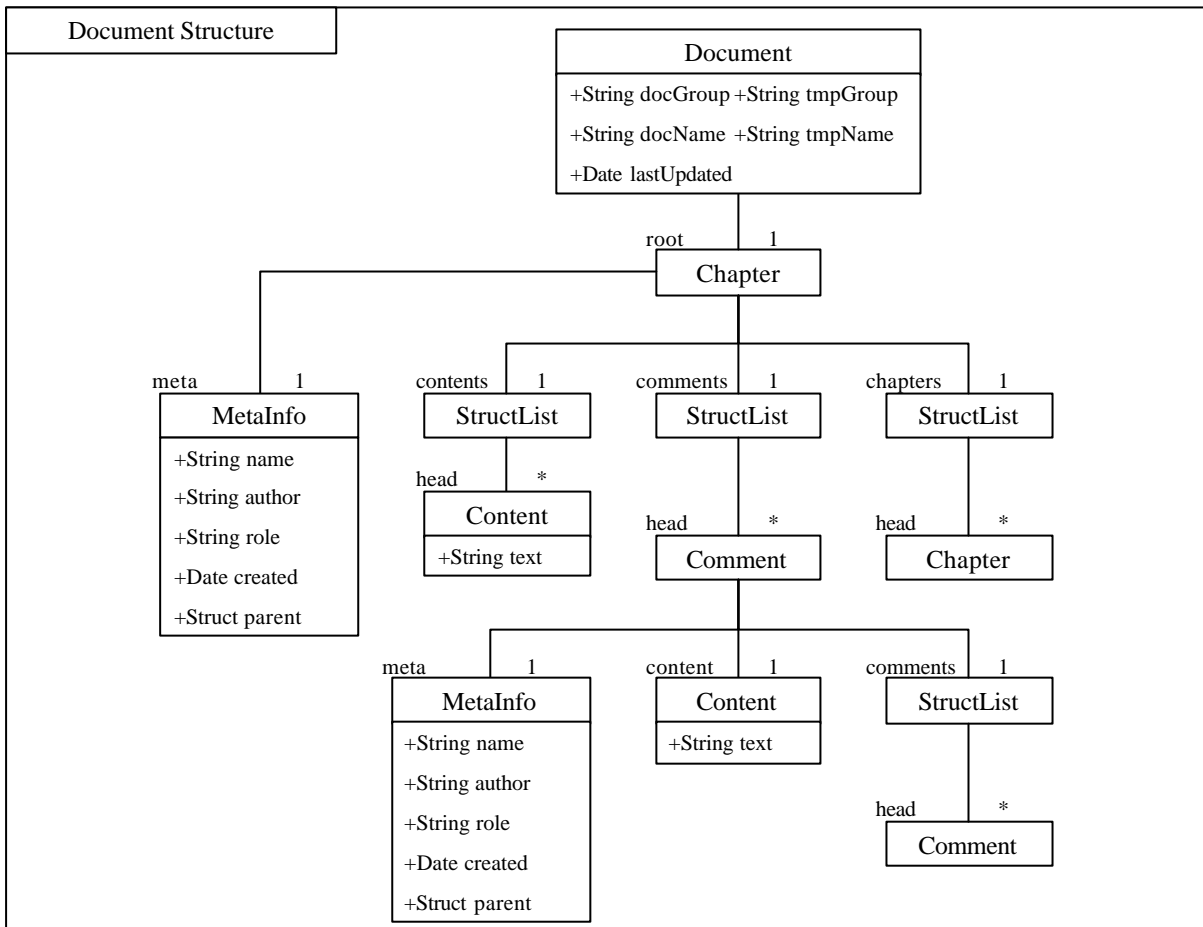


Figure 4-5. Document structure

Figure 4-6 illustrates how the system generates lists of document (or template) metadata using class StructList.

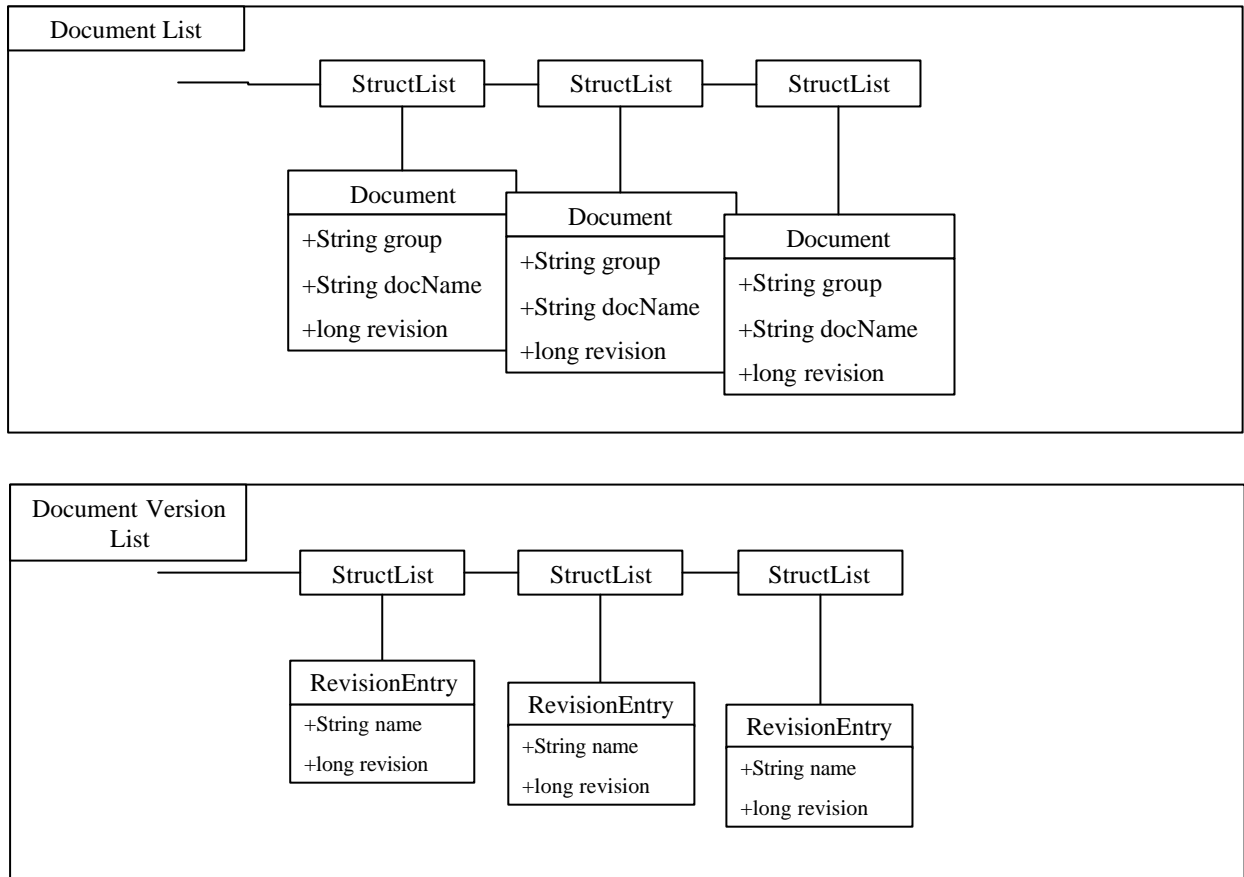


Figure 4-6. Document or template lists

4.2.3 Templates

The structure of a template in the DECA system is shown in figure 4-7. A template is similar to a document, with the exception of not having any comments attached.

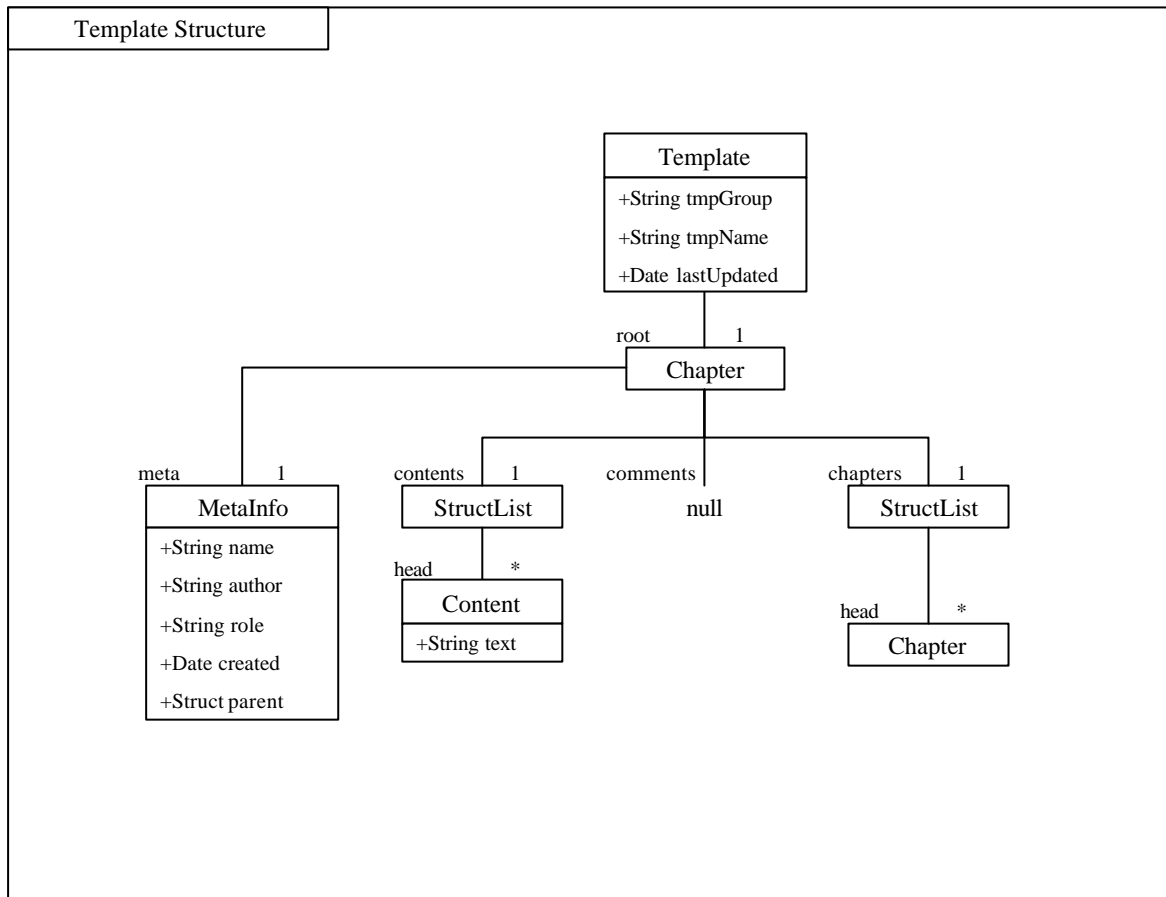


Figure 4-7. Template structure

4.2.4 Version and access control

The version and access control structures used in the DECA system are represented in figure 4-8.

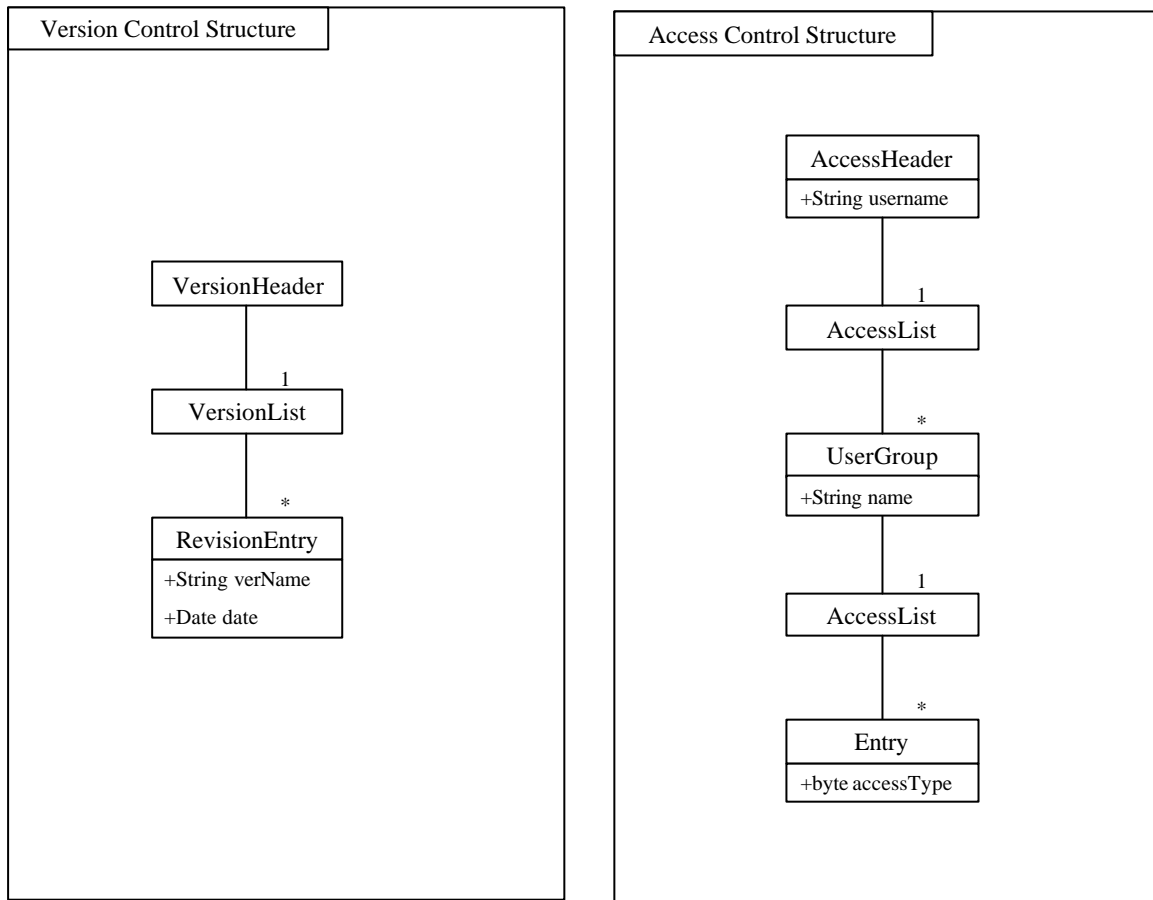


Figure 4-8. Version and access control structures

4.2.4.1 Version control

Version control structure in the DECA system assigns each document or template a list of all the version labels assigned to it. Each version label specifies a certain moment in time, which can be labeled.

4.2.4.2 Access control

For every user there is an AccessHeader which contains a list of UserGroups the user belongs to. Every UserGroup defines a list of Entry-objects, which relate to one document or template and define access type for it.

5. ERROR HANDLING

5.1 Overview

Error handling in the DECA system is based on user action logging. All the recoverable Java exceptions are caught in the code and logged correspondingly. The user will be notified if the requested action couldn't be executed.

If an exception somehow passes the DECA code into the JSP-engine, a message stating that a fatal error has occurred is shown. In addition, the session-specific user log is displayed on screen for debugging purposes.

5.2 Error types

5.2.1 Fatal errors

The software has two types of errors. The first type is errors that are "fatal" to the DECA system. In these cases the execution cannot continue. These types of errors are usually generated by software malfunction.

5.2.2 Non-fatal errors / warnings

Non-fatal errors are errors that a system can tolerate. The situations where these kinds of errors may occur are described in [funcspec].

5.3 Logging

The DECA system uses two-stage error and debug logging. Tomcat web server itself generates `jvm.stderr` and `jvm.stdout` log files for `stderr` and `stdout` streams. These files contain the debug output and all the errors of all the users in the system.

Second stage logs are gathered per session. User actions generate debug log into a stream. This stream is saved to a predefined directory on a web server for later use. Old logs can be purged from the directory with a separate scheduled task.

6. DISCARDED APPROACHES

6.1 Relational database as storage format

6.1.1 Reasons for discarding

The main functionality of the DECA system is based on handling massive tree-like data structures. Designing and implementing a system that stores/retrieves object trees to/from standard relational databases requires a large amount of work.

Sections 6.1.2 – 6.1.4 represent the discarded database design based on relational databases.

6.1.2 Versioning principles

Versioning is implemented by giving *in* and *out* timestamps to each document element. *In* timestamp defines the moment when the element becomes valid. Correspondingly, *out* timestamp defines the invalidation time of the element. This approach makes it possible to fetch documents (including comments) as they were in any given time.

Internally the technique described above is used only on contents and comments (see section 3.2.5).

6.1.3 Representing trees in SQL tables

The usual way to represent trees in SQL is to use the adjacency list model (illustrated in figure 3-3), which is described in most of the SQL books. This method has several disadvantages in an SQL DBMS which doesn't have support for tree handling. Queries to an adjacency list must be recursive. Many nested select-statements are inefficient especially in deep trees. Adjacency lists also contain duplicate data, which is waste of space.

Table Personnel:

emp	boss	salary
'Albert'	'NULL'	1000.00
'Bert'	'Albert'	900.00
'Chuck'	'Albert'	900.00
'Donna'	'Chuck'	800.00
'Eddie'	'Chuck'	700.00
'Fred'	'Chuck'	600.00

Figure 3-3. An example of an adjacency list

Better way to represent trees is to show them as nested sets. Since SQL is a set oriented language, this is a better model than the usual adjacency list approach. Nested set model adds special 'left' and 'right' values to each node. This approach is illustrated in figure 3-4.

[will not be implemented due to discarding]

Figure 3-4. Creating a nested set.

To convert a tree into a nested sets model think of a squirrel moving along the tree. The squirrel starts at the top, the root, and moves anti-clockwise along the tree. When it comes to a node, it assigns a value in the cell on the side that it is visiting and increments its

Luomi, Rautopuro, Öhman

25.03.2002

counter. Each node will get two values, one of the right side and one for the left. This is known as modified preorder tree traversal algorithm.

6.1.4 Database tables

6.1.4.1 Table document

Table *document* contains information about the documents and templates stored in the system.

Table *document* is created with the following SQL clause:

```
create table document (  
    do_id                smallint unsigned not null,  
    do_space             smallint unsigned not null,  
    do_path              lvarchar not null,  
    do_author            varchar(30) not null,  
    do_version           timestamp,  
    do_label             varchar(50) not null,  
    primary key(do_id, do_version)  
);
```

Description of each column follows. Primary key columns are represented with asterisks.

do_id *	Identifies the document or the template.
do_space	Attaches a structure to this document. The structure of the document is specified in table <i>structure</i> .
do_path	Specifies the name of the document and its position in the document hierarchy. For example, document 'market review' in document group 'sales/division a/mr. johnson' would be represented with path 'sales/division a/mr. johnson/market review'.
do_author	The author of the document.
do_version *	A document version is identified by a timestamp, which is the time when the document version was 'freezed'. The most recent version of the document is identified by a NULL timestamp.
do_label	A label of the document version which contains version-specific information.

6.1.4.2 Table structure

Each document and template described in table *document* has a structure. These structures are represented in table *structure*. A structure is a tree, which is stored in a special form described earlier in section 3.2.3.

Table *structure* is created with the following SQL clause:

```
create table structure (  
    st_id                smallint unsigned,  
    st_space             smallint unsigned not null,  
    st_name              varchar(50) not null,
```

Luomi, Rautopuro, Öhman

25.03.2002

```

        st_left          smallint unsigned not null,
        st_right         smallint unsigned not null,
        primary key(st_space, st_left)
    );

```

Description of each column follows. Primary key columns are represented with asterisks.

st_id	Identifies the structure element. Set to NULL, if this element can't contain anything.
st_space *	Uniquely identifies the structure.
st_name	Name of this structure element (for example, a title of a paragraph).
st_left *	Tree representation. (See section 3.2.4.)
st_right	Tree representation. (See section 3.2.4.)

6.1.4.3 Table content

A structure element can contain only one content. A content can be binary or ASCII data. A content is versioned by the method described in section 3.2.3.

Table *content* is created with the following SQL clauses:

```

create table content (
    cn_id          smallint unsigned not null,
    cn_text        text not null,
    cn_in          timestamp not null,
    cn_out         timestamp,
    primary key(cn_id, cn_in)
);

```

Description of each column follows. Primary key columns are represented with asterisks.

cn_id *	Identifies the structure element in which this content belongs to.
cn_in *	Versioning. (See section 3.2.3.)
cn_out	Versioning. (See section 3.2.3.)
cn_text	The content data.

6.1.4.4 Table comment

A single structure element can have a tree of comments. A comment can be binary or ASCII data. A comment is versioned by the method described in section 3.2.3.

Table *comment* is created with the following SQL clauses:

```

create table comment (
    co_space       smallint unsigned not null,
    co_left        smallint unsigned not null,
    co_right       smallint unsigned not null,
    co_in          timestamp not null,
    co_out         timestamp,

```

Luomi, Rautopuro, Öhman

25.03.2002

```

        co_text          text not null,
        primary key(co_space, co_left, co_in)
    );

```

Description of each column follows. Primary key columns are represented with asterisks.

co_space *	Identifies the structure element in which this comment belongs to and uniquely identifies the comment tree.
co_left *	Tree representation. (See section 3.2.4.)
co_right	Tree representation. (See section 3.2.4.)
co_in *	Versioning. (See section 3.2.3.)
co_out	Versioning. (See section 3.2.3.)
co_text	The comment data.

6.1.4.5 Table user [under construction]

Table *user* is created with the following SQL clauses:

```

create table user (
    us_id          varchar(30) not null,
    us_role        varchar(50) not null
    primary key(us_id)
);

```

Description of each column follows. Primary key columns are represented with asterisks.

us_id *	User id.
us_role	User's role.

6.2 XML as storage format for documents

We could not see the benefit of storing the data as XML fragments in the database. Using XML in the DECA project would need code to encode and decode the data into XML. We found a simpler approach: to store the data into the database by mapping data structures into database rows. One instance of a data structure would convert easily into a single database row. We could output the documents as XML, if we programmed a function to convert our data into XML. Such programming would be easy but time-consuming.

Luomi, Rautopuro, Öhman

25.03.2002

7. OPEN ISSUES

7.1 User interface issues

The specification of the user interface taglet classes will be revised, when the first round of user interface testing is complete and the results have been analyzed. However, general ideas represented in this document are ready to be used in implementation testing.

7.2 Missing class descriptions

The specification of the filter classes will be added after they are implemented. Other class descriptions, such as general utility classes, will be revised or written more detailed during the implementation phase.

7.3 User interface specification

The user interface specification is still missing. This problem will be fixed in phase T3.