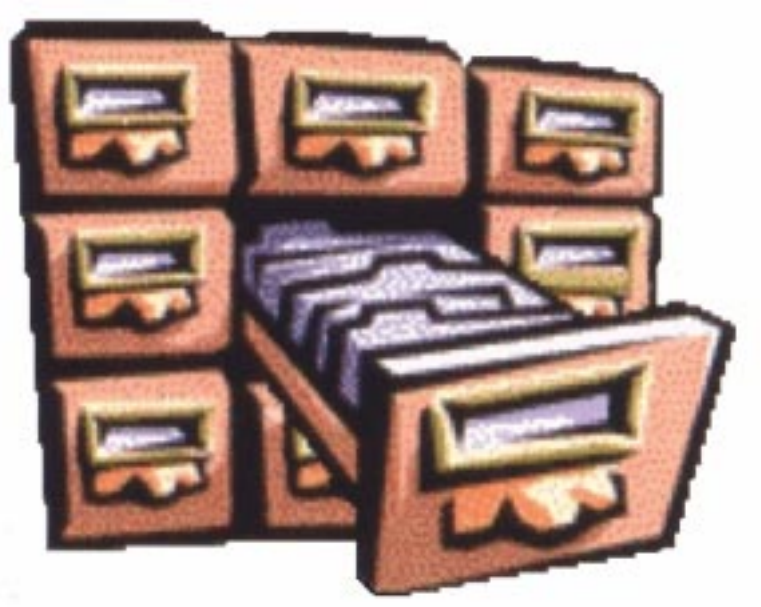
Architectural Design Document



The eArchive

The archive drawer of the future

Version 3.0

Status:	Externally approved
Authors:	Steven Hoebens, Mark Scheffer
Internal Review:	Erik Luit, Hugo Jonker
External Review:	Lou Somers, Tom Verhoeff, Erik Luit

Architectural Design Document

Project: eArchive

Project Group: SEeA

Signature Senior Management:

_____

(Dhr. E.J.Luit, Eindhoven University of Technology)

July 2000
Eindhoven University of Technology

a. Document Status Sheet

DOCUMENT STATUS SHEET			
Document Title: Architectural Design Document			
Document Authors: Steven Hoebens, Mark Scheffer			
Document Identification: SEeA/Documenten/ADD/ADD300.doc			
Document Status: concept / internally accepted / **externally approved**			
ISSUE	**REVISION**	**DATE**	**REASON FOR CHANGE**
0	00	21-02-2000	Creation of document
0	01	09-02-2000	Extension of contents
0	02	05-04-2000	Extension of contents
0	03	11-04-2000	Figures changed to UML standard
0	04	13-04-2000	Extension of contents
1	00	13-04-2000	Document for external review
1	01	17-04-2000	Comment of external review 17-4-2000
2	00	21-06-2000	Created new ADD, reusing parts of previous versions
2	01	01-07-2000	New models added for chapter 4, traceability matrix added
2	02	03-07-2000	Fixes after review of Erik Luit
2	03	05-07-2000	Improved structure and interfaces, updated chapter 7
2	05	06-07-2000	Minor errors, improved layout
3	00	06-07-2000	Spelling check and label ‘externally approved’

b. Document Change Record

DOCUMENT CHANGE RECORD	
Document Title: Architectural Design Document	
Document Identification: SEeA/Documenten/ADD/ADD300.doc	
Section	**Summary of Changes**
All	Complete rewrite of ADD to reflect architectural design of the product

c. Abstract

This document records the Architectural Design (AD) for the eArchive Software Engineering project. This projects purpose is to research the possibilities for an electronic archiving system for Océ Technologies B.V. (Océ).
The Architectural Design Document (ADD) describes the architectural design using the software requirements and the logical model defined in the Software Requirements Document [SRD].
The responsible team, the Architectural Design Team consisted of Bart van der Meulen, Steven Hoebens, Mark Scheffer, Arno Ansems and team leader Jan van Thiel.
Steven Hoebens and Mark Scheffer rewrote the Architectural Design Document from version 2 onwards.
The document complies with the standards for an Architectural Design Document (ADD) from the Software Engineering Standard, as set by the European Space Agency [ESA].
The models described in this document will only reflect the basic architecture of the system. Further development of the components is done in the Detailed Design Phase.

d. Table of Contents

1. Introduction

This section describes the purpose and the scope of this document. Also the definitions, acronyms and abbreviations used in this document are described in this section. The last two parts of this section contain references and an overview of the entire document.

1.1. Purpose

The ADD provides a beginning of a solution - by means of a component decomposition - for all software requirements as specified in the Software Requirements Document ([SRD]) and defines a basic object model specifying the components and interfaces between components within the User Interface and the eArchive program which are developed during the project. This document is intended to be read by all members of the project, as well as by the senior management.

1.2. Scope

The eArchive project is an assignment of the course Software Engineering (2R690) at the Eindhoven University of Technology (TUE). Océ Technologies supplied the assignment. Purpose of the project is to develop an electronic archive system. The ADD describes the Architectural Design for the User Interface and the eArchive program.

The eArchive program design and construction is intended as a pilot project for the Research and Development Department of Océ B.V. (Océ). Océ wants to assess the possibilities that are available with today's techniques to create an electronic archive. The program is designed using an object orientated design method. For further description of the system and the background see [URD, chapter 2] and [SRD, chapter 2]. More information about the context of this project can be found on the directory */context* of the eArchive installation cdrom.

1.3. Definitions and abbreviations

This section provides a list of definitions and a list of abbreviations used in this document. Both lists are sorted in alphabetic order. They are for a large part copied from the [URD] and [SRD], where some of the explanations may be more elaborate than they are here.

Definitions:

Document Classification	The division of documents into classes
Document Life Cycle	The ordered list of *statuses* a document will pass through.
Document Representation	(Form of) appearance of the document
Document Status	The state of a document in an archive. The *status* of a document can dictate which storage method is used and what the life span of the document is.
Meta-document	*Metainformation* of an archived document.
Metainformation	A collection of data describing the content and context of a document.
Reference	Link of a document to another document that is already present in the archive. Such a reference is directed, i.e. if document ‘A’ is linked with a reference to document ‘B’, then a link is not (automatically) included from ‘B’ to ‘A’.

	Note that this does not mean that two documents cannot have references referring to each other (cyclic references are allowed). References are a dynamic part of the *metainformation*.
System	The electronic archive (eArchive)

Abbreviations:

ADD	Architectural Design Document
DBMS	DataBase Management System
DDD	Detailed Design Document
ESA	European Space Agency
HTMLG	HTML Generator
IM	Information Manager
JDBC	Java Database Connection
LIU	Logged in User
SE	Software Engineering
SEeA	Software Engineering eArchive
SRD	Software Requirements Document
SysOp	System Operator
URD	User Requirements Document

1.4. References

[ESA]	ESA Software Engineering Standards, ESA PSS-05-02, Issue March 1995, ESA Board of Software Standardisation and Control (BSSC), ISBN 0-13-106568-8
[DCS-ER]	Design Criteria Standard for Electronic Record Issue November 1997 Department of Defence Management Software Applications, DoD 5012.2-STD,
[DoD]	Subset of [DCS-ER], denoted by the marked parts R. Kersemakers, January 2000 Config-ID: `Archive/Oce/Classifications/dod.pdf`
[SC]	Standard classifications, R&D Internal Document Types, Information Management department, 17-12-1999, Document classification of Océ R&D Config-ID: `Archive/Oce/Classifications/document types within R&D.pdf`

[URD] User Requirements Document
SEeA URD-team,
January 2000,
Config-ID: `/Master/URD`

[SRD] Software Requirements Document
SEeA SRD-team,
January 2000,
Config-ID: `/Master/SRD`

[UML] De UML toolkit,
Hans-Erik Eriksson, Magnus Penker,
Academic Service,
ISBN 90 395 1015 6

[OMT] College Dictaat,
Software Engineering 2M390,
Eindhoven University of Technology

[MySQL] MySQL Reference Manual for version 3.23.10-alpha
Detron HB and Monty Program KB
Config-ID: `Archive/Components/Database/MySQL.pdf`

[JDBC] Java DataBase Control Software 1.2C
Mark Matthews
GNU GPL
http://java.sun.com/jdbc/

1.5. Overview

This ADD contains a decomposition of the software to be developed into components and a detailed description of each component.
Chapter 2 of this document is a short introduction to the system context and background of the project, and briefly discusses the system design decisions. Also included is a short explanation about the function and working of servlets. In chapter 3 the relationship with external systems is described. In chapter 4 the decomposition of the system in smaller components is described. Each of these components is specified in chapter 5. Chapter 6 contains an estimate of the resources needed to build and maintain the system. In chapter 7 a traceability matrix that links each [SRD] requirement to a component described in chapter 5 is given.

2. System Overview

This section briefly discusses the background, system context and basic top-level design. The architectural design is further documented in chapter 3.

2.1 Background of the system

The function and purpose of this project are described in [URD], paragraphs 2.1, 2.2, and 2.4.
The environment in which the system runs is described in [URD], paragraph 2.5.
More information about the context and relations to other systems can be found on the directory */context* on the installation CD or in the [URD], paragraph 2.6.

2.2 Basic design and context

At the top level of the system, we can distinguish three major parts. These are the server, the client and the store (fig. 1). A logical model of the eArchive is constructed in the [SRD]. This logical model mainly describes the behaviour of the server part of figure 1. The client is described in the prototype [SRD, section 4.1]. The store is a part of the system that consists of a storage facility (most likely to be hard disk) that is not provided by the SEeA team. The store contains a storage facility to save information in an external database (MySQL) and a file system. Most commonly, the store is implemented by a hard disk on the server computer.

In the next sections this model will be refined with an explanation about the selected architecture.
For detailed information about the system context and design, please regard chapters 3 and 4 of this document. The detailed design is documented in the [DDD].

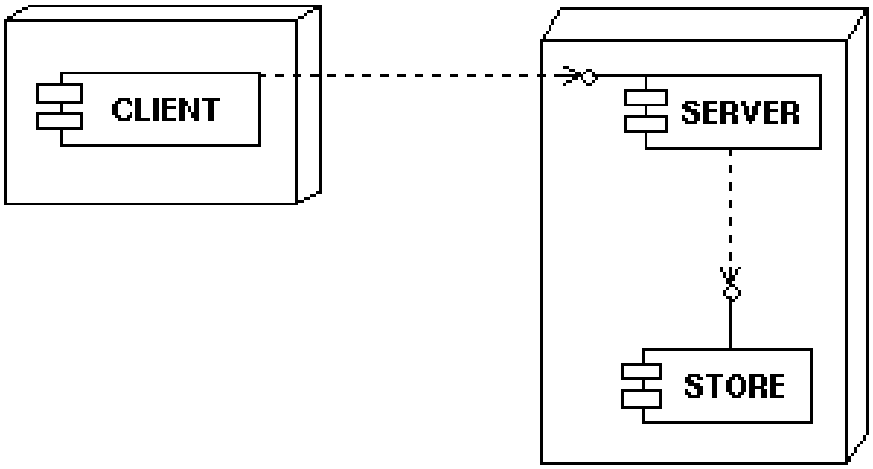


Figure 1: ***Three major components***

2.3 Design Decisions and Preliminary information: Servlets

2.3.1 Servlet

Servlets, why use them?

We have a client system that sends requests to the system (see figure 1). Servlets are modules that extend request/response-oriented servers, such as Java-enabled web servers. Therefore a servlet is a logical choice to extend the request/response-oriented Apache server. The servlet is responsible for taking data in an HTML form or link and applying the eArchive logic to update the store if necessary and sent the results back to the browser. Servlets also deal with concurrency and provide session management.

Servlets provide a way to generate dynamic documents that is both easy to write and fast to run. Servlets address the problem of doing server-side programming with platform-specific APIs: they are developed with the Java Servlet API, a standard Java extension. Since the rest of the server site software also is written in Java, the Java Servlet API perfectly fits in.

It is easy to use servlets to handle HTTP client requests. For example, we can have servlets process data POSTed over HTTPS using an HTML form, including metainformation or credit card data. A servlet like this can be part of the eArchive system, working with a database, and perhaps an on-line payment system.

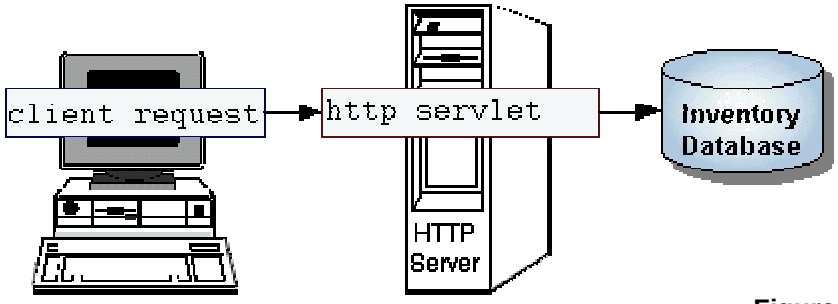


Figure 2: *HttpServlet handles requests*

Servlets, how do they work?

The javax.servlet package provides interfaces and classes for writing servlets. The class HttpServlet inherits from a class GenericServlet. The central abstraction in the Servlet API is the Servlet interface. All servlets implement this interface by extending a class that implements it such as HttpServlet. The Servlet interface declares, but does not implement, methods that manage the servlet and its communications with clients.

Client Interaction

When a servlet accepts a call from a client, it receives two objects:

- A ServletRequest, which encapsulates the communication from the client to the server.
- A ServletResponse, which encapsulates the communication from the servlet back to the client.

ServletRequest and ServletResponse are interfaces defined by the javax.servlet package.

A user operates the system by using a web browser. The user can perform actions by entering information and clicking on links of the HTML-interface, provided by the web browser. The web browser then sends certain information with regard to the user action to the (web) server (a servlet). The web server starts a session for each user connection.

A so-called (Http)ServletRequest and (Http)ServletResponse form the exact information that is sent from the client (web browser) to the server. Together they form a standard object that is sent by web browsers. It includes a sessionId and parameters (contained in ServletRequest). With each request, a parameter PageID is sent that indicates the page that is requested. Such a ServletRequest thus provides a unique id for each session. Thereby it is known which user request belongs to which browser. A string identifies the page that is requested by the user. If there are any attributes involved with that request, they are also part of the ServletRequest.

The *ServletRequest* interface allows the servlet access to:

Information such as the names of the parameters passed in by the client, the protocol (scheme) being used by the client, and the names of the remote host that made the request and the server that received it.

The input stream, ServletInputStream. Servlets use the input stream to get data from clients that use application protocols such as the HTTP POST and PUT methods.

The *ServletResponse* interface gives the servlet methods for replying to the client. It:

Allows the servlet to set the content length and MIME type of the reply.

Provides an output stream, ServletOutputStream, and a Writer through which the servlet can send the reply data.

Servlets have no graphical user interface and can be embedded in many different servers because the servlet API, which is used to write servlets, assumes nothing about the server's environment or protocol. Servlets have become most widely used within HTTP servers and many web servers support the Servlet API. HTTP servlets have some additional objects that provide session-tracking capabilities. We have used these APIs to maintain state between the servlet and the client that persists across multiple connections during some time period.

2.3.2. Linux Operating System with Apache server

The operating system for the system is Linux, because it is freely available, open source, very stable and Océ already has experience working with Linux.
We have chosen to use Apache as web server because it is widely used, it has the ability to run Servlets and because it is freely available. Apache is also known as a stable web server and supports secure sockets layer (see chapter 2.3.5. for more information).

2.3.3. Store and JDBC Control

JDBC Control - What is it and why have we chosen to use it?

Because we want to abstract from the internal structure of the store, all requests to and from the store are handled via a component Java DataBase Connection (JDBC) Control. Thus to abstract from the storage method, the JDBC Control component is used to handle all communication with the store. If the method of storage would change (for example if a different database would be used), this would only have impact on the JDBC Control component.

Example: a request for information from the store or an update command on the store is sent to JDBC Control. The JDBC Control component then handles the request by passing on the SQL queries to the JDBC interface. It is thus an abstract layer that provides an interface to the store.

JDBC is a Java API that consists of a set of classes and interfaces written in the Java programming language. JDBC provides a standard API for tool/database developers and makes it possible to write database applications using a pure Java API. Using JDBC, it is easy to send SQL statements to virtually any relational database. In other words, with the JDBC API, it isn't necessary to write one program to access a MySQL database, another program to access an Oracle database, another program to access an Informix database, and so on. We can now only use a single program using the JDBC API, and the program will be able to send SQL statements to the appropriate database. And, with an application written in the Java programming language, one also doesn't have to worry about writing different applications to run on different platforms. The combination of Java and JDBC lets us write it once and run it anywhere.

MySQL Database

Metadocuments, digital representations, user information and their attributes (user groups, access rights, etc.) are kept in the store. According to the [SRD], it must be possible to find metadocuments by searching on elements of metainformation. We have chosen to use a database to store these items. This way searches can be implemented using the server to query the database (fig. 2). We need a freely accessible database; it has been decided to use the open source DBMS MySQL. MySQL is accessed by the Server component with a JDBC; see paragraph 4.2.3 for details.

The data is stored in separate tables; the relations between these tables are defined by the DataClasses component of the Server Component (see 4.2.3).

Folder Structure

The digital representations of documents must be accessible without using the database. This is a security requirement of Océ. Therefore all digital representations are stored in a normal folder structure (fig. 2). For this folder structure the same operating system as the server has been chosen: Linux. This because it is imaginable that the Store and the Server reside on one system; to accommodate this the same operating system as the server is required.

2.3.4. HTML Generator

Users can access the system via web pages. These web pages are defined in the prototype [SRD, section 4.1]. The choice has been made to send complete HTML-pages to the client. This way the logical model will not be divided over the server and the client. This also means that the system runs completely on the server. Every time the user performs an action the server will generate a new HTML-page. This is done by the HTML Generator component.
Another option we considered was a client application that communicates with the server. This option is more powerful than the option we have chosen, but is also far more complicated (since the system also has to be zero-install to comply with SR201 (see [SRD])). Using only HTML-pages as the interface on the client side also has the advantage that the system requirements for the client system are very simple; only a web browser is needed.
We have extended the standard html pages with JavaScript to make certain changes in the html pages, because it would take too much time to download a complete html page from the server for all actions. JavaScript can for example change the available metainformation after the documenttype is selected.

2.3.5. Encryption

In case we use the system in an external setting, the communication between the client and the server is encrypted. We decided to use SSL (secure sockets layer) as encryption method. SSL is an encryption technology that scrambles a message so that only the recipient can unscramble it, using technologies developed by RSA security. URLs that begin with "https://" are using SSL.

Why did we choose to use SSL?

The eArchive system is unaware of this secure connection. So for the system, SSL provide a simple solution to the security requirements (see [SRD] requirement SR601).
To implement this kind of encryption technology, we need to enable SSL on our Web server. Apache supports this method of encryption, using SSL. But even if SSL is on the server, it'll only work with other SSL-friendly browsers. SSL is available though in both Netscape 4.5, or higher, and Internet Explorer 4, or higher (conform the assumptions from chapter 2.6 [URD]).

3. System Context

In this section the external interfaces of the system are described.

The URD[2.5] and SRD[2.3] also contain some information about the system context.
We distinguish two different external interfaces of the system and draw up an inventory of what the minimal information is that needs to be exchanged with the system.

Between Client and Servlet

When the user sends a request via a web browser, the servlet receives two objects:

- A ServletRequest, which encapsulates the communication from the client to the server.
- A ServletResponse, which encapsulates the communication from the servlet back to the client.

When the system gets a request, it sends back a new HTML-page to the browser, using the -outputstream of ServletResponse.

Between System and Store

As we can see from the URD, the store is external to the system. In the store metadocuments and digital representations are stored. For that purpose, the store contains a database and a folder structure respectively.

Requests to the database are translated into database queries by JDBC Control by asking the DBMS (MySQL is the database that is used) to send the results of the query. The transfer of queries and results is finally handled by the MySQL driver (org.gjt.mm.mysql).

The other part of the store, the folder structure, stores digital representations. Digital representations can be directly transferred to and from the folder structure. The com.oreilly.servlet driver facilitates these uploads and downloads of files.

The interface between the system thus consists of two parts:

- via JDBC Control (and more directly the MySQL driver org.gjt.mm.mysql) queries for the database are sent to the MySQL database. The results of the query are then sent back. A query or query-result might contain (information about) metadocuments.
- digital representations can be directly saved and retrieved from the folder structure.

For more information about the interface between system components and the store, we refer to chapters 5.7.6 (JDBC), 5.9.6. (Database) and 5.6.10 (FolderStructure).

4. System Design

This section describes the method used for the decomposition of the system and gives a short description of the system components.

4.1. Design Method

The method used to design the component model is [OMT]. The model is made with [UML]. The presented model is a decomposition view into components. A top-down approach is used to decompose the system.

4.2. Decomposition description

At the top level of the system, as mentioned in figure 1, three major parts can be distinguished. These are the server, the client and the store (fig. 1).

4.2.1. Client

Users should be able to operate the system using a web browser (see also [URD] chapters 2.5 and 2.6). The output of the system thus consists at least of Hypertext Mark-up Language (HTML) files, that are interpreted by web browsers. The clients communicates with the system using a web browser; the web browser in his turn communicates with the system using a Servlet. The interface between the client and servlet is further described in chapter 3. More information about servlets can be found in chapter 2.1. The client is able to communicate with the servlet via the Apache web server (see also paragraph 2.3.2. and figure 2).

4.2.2. Store

The store is divided into two components: the Database and the FolderStructure (see 5.9, 5.10). Metadocuments, digital representations, user information and their attributes (user groups, access rights, etc.) are kept in the store. These storables are stored in database and in a folder structure. More information about this decision design can be found in chapter 2.3.3. The Database is implemented by MySQL; the server connects to MySQL using the JDBC interface. The data is stored in separate tables for each type of storable; the relations between these tables are defined by the DataClasses component of the Server Component (see 4.2.3). Chapter 3 contains a description of the interface between the store and the system.

Not all data will be stored in the Store. This has two reasons. The first reason is that some data is needed frequently and a local copy is kept for efficiency reasons. The second reason is that some data does not need to be stored in the Store. For example, it is unnecessary to keep data about an active session in the store.

4.2.3. Server

In figure 2 we can see the Server and (connection to) the Client and Store. The architecture that remains on the Server side should make sure all requests are handled by performing the needed updates on the store and by creating a result html page and sent it back to the client.

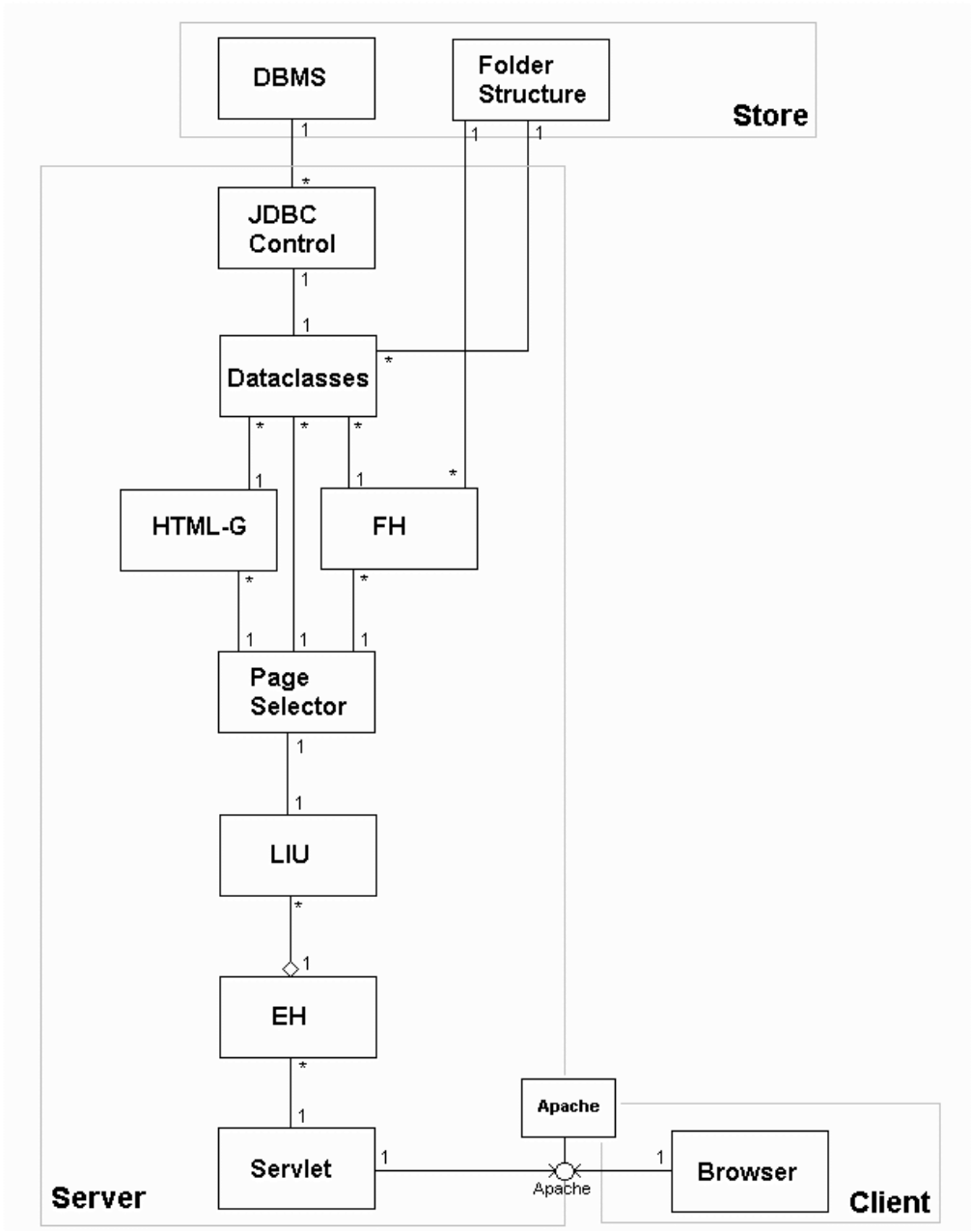


Figure 2: *Structure of the server components*

The most important abbreviations of figure 2: EH: Event Handler, LIU: LoggedInUser, FH: FormHandler, HTML-G: HTML Generator.

In order to do that, the server of course needs to know to which user to sent the result html page. Another requirement is that all requests should be received immediately; a second request should not have to wait until the first request is completed. This could cause unnecessary delay (for example when the first request requires several queries on the database that take relatively much time to complete).

All communication between the client browser (see 4.2.1) and the server is handled by the standard component EventHandler. Because the server somehow needs to keep track of which users are using the system, we introduce a LoggedInUser object, which is responsible for managing user actions.
This way we have several LoggedInUsers, each responsible for managing requests of their corresponding (logged in) users. All LoggedInUsers can process their own requests in parallel. The EventHandler keeps track of the information needed to sent all incoming requests to the right LoggedInUser.

Would should be done for each request? First we should check with what kind of request we deal with. This is done by the component PageSelector. This component determines what updates on the database are necessary and what kind of result html page is needed.

A request can involve updates of the database content, for example when a user submits a new metadocument. The component FormHandler is responsible for these updates. With each request a html page should be returned to the browser. We strive for separation of concerns, and use another component to return the result pages: HTML Generator is responsible for creating these html pages and returns them to the client browser.

With this knowledge, we know regard the whole process:

When a user clicks on a link or button on an HTML-page an HTML-request is sent to the Server. This request is intercepted by the Event Handler. The Event Handler will determine from which user the request came, and will send it to the appropriate LoggedInUser.

LoggedInUser in turn passes it on to the PageSelector, which further interprets the content of the request and passes update commands to the FormHandler component should an update be required; the PageSelector passes page requests to the HTML Generator component. The PageSelector and HTML Generator components pass update commands and read queries to the DataClasses component.

The DataClasses component represents the data in the Store: it passes updates and queries on this data through the JDBC Control component to the Database. The Folder and Digital Representation data classes also update and query the folder structure of the store; this way DataClasses functions as the object layer for the storables in the store. Within these data classes the relations between the different storables (see [SRD]) is defined (fig. 3, 4, 5, 6). These classes and connections coincide with the relations from the [SRD].

These are the exact classes (in alphabetical order) the component DataClasses consists of:
AccessRights, Classification, DigitalRepresentation, Folder, ID, MetaDocument, MetaInfo, MetaType NextStatus, Parser, PhysicalRepresentation, Status, UserGroup, UserInfo, UserPrefs, SetOfClassification, SetOfDigitalRepresentation, SetOfFolders, SetOfMetaDocuments, SetOfMetaTypes, SetOfMetaInformation, SetOfPhysicalRepresentations, SetOfStatuses, SetOfUserGroups, SetOfUserInfos, FormHandler.

The further design of the server components is reasonably trivial – except for the dataclasses. Since a similar design structure is already given in the [SRD], more simplified models are presented here, a detailed description of these models will be given in the [DDD]. The models here indicate the relations between the different elements of DataClasses.

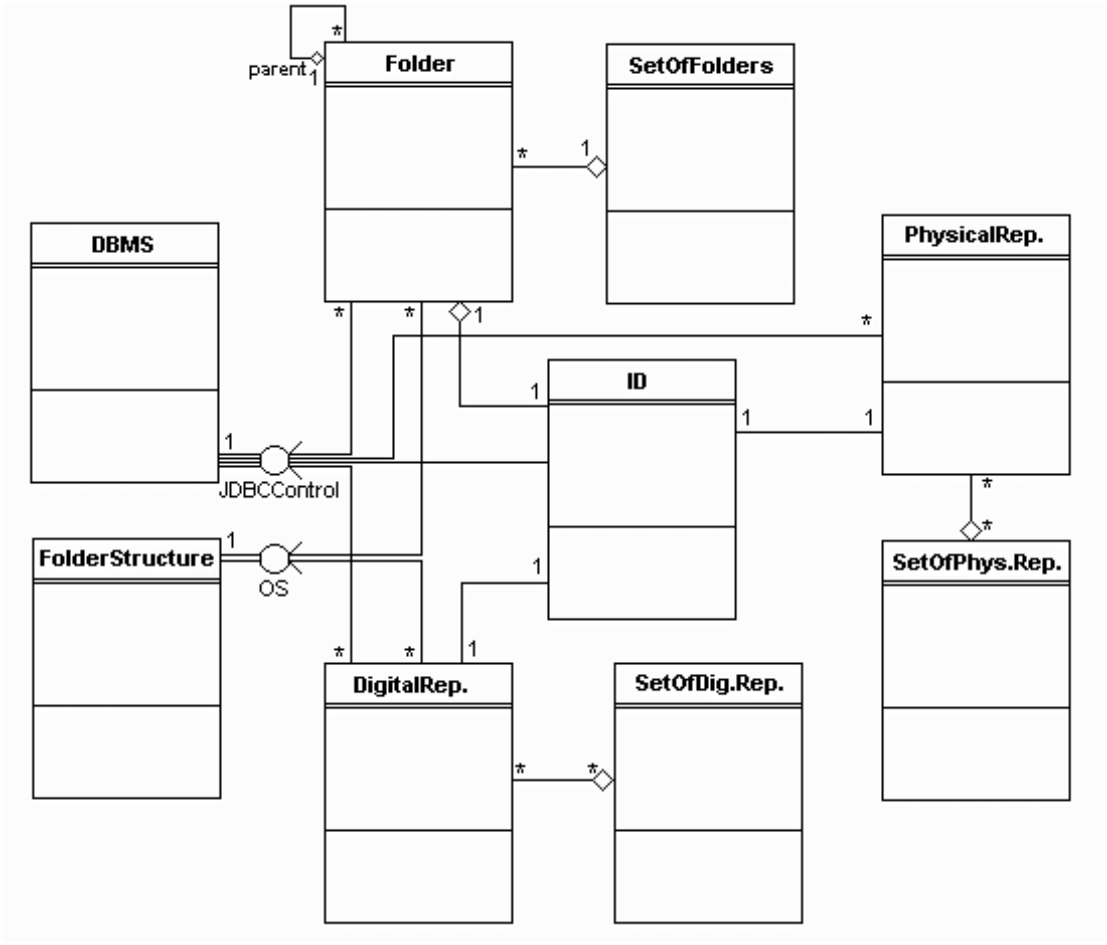


Figure 3: *Folder, Representations and ID*

Most notable is the introduction of the ID class. This class is responsible for giving off unique identifiers to new storables. This class has been introduced to abstract the identities in the database.

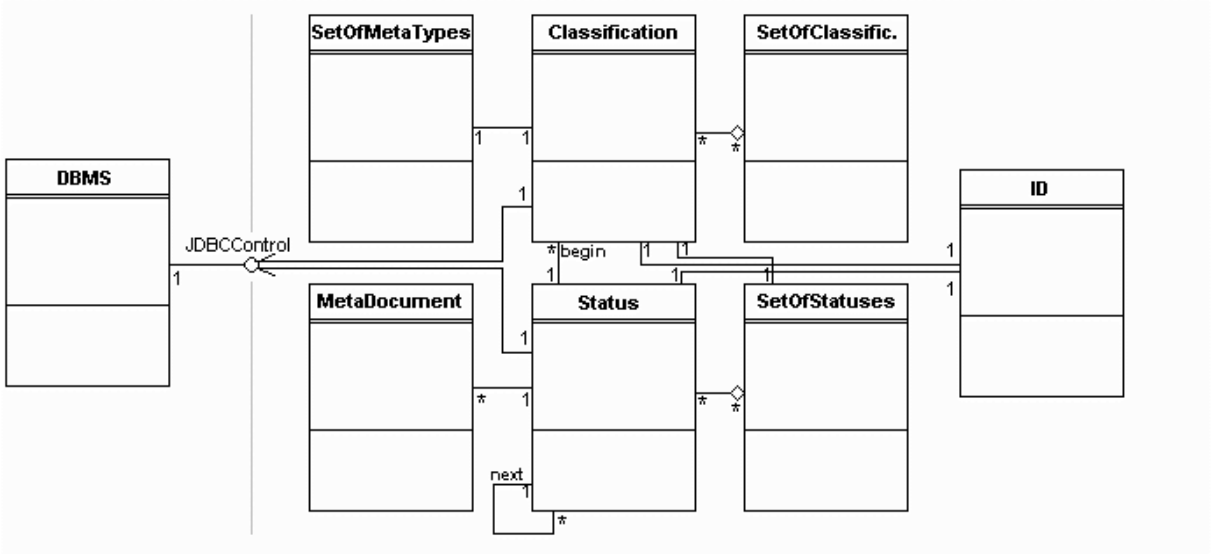


Figure 4: *Classifications and Statuses*

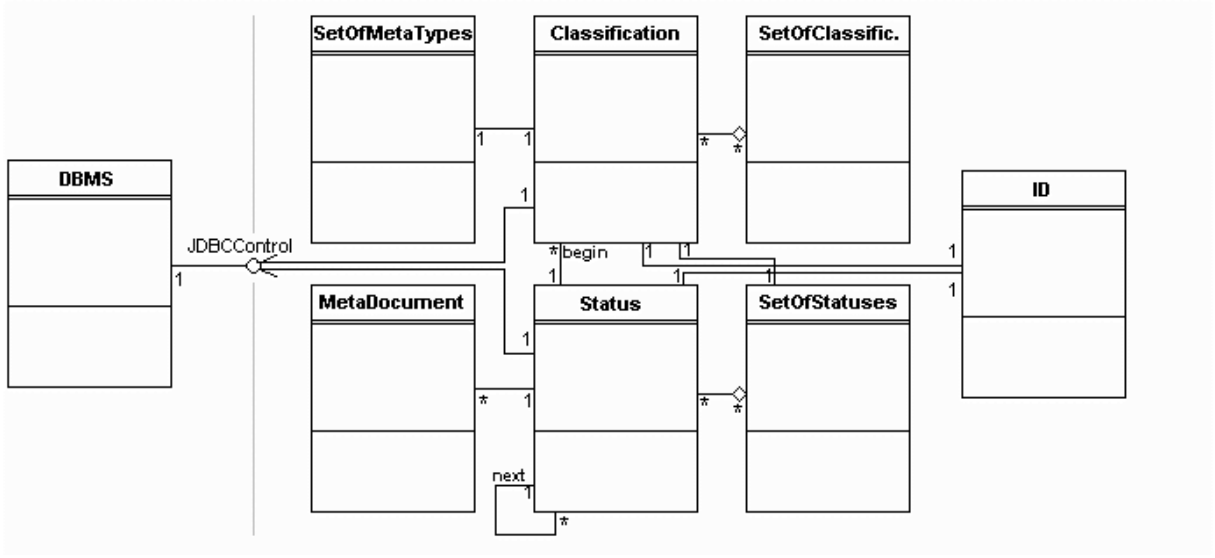


Figure 5: *MetaDocument and AccessRights*

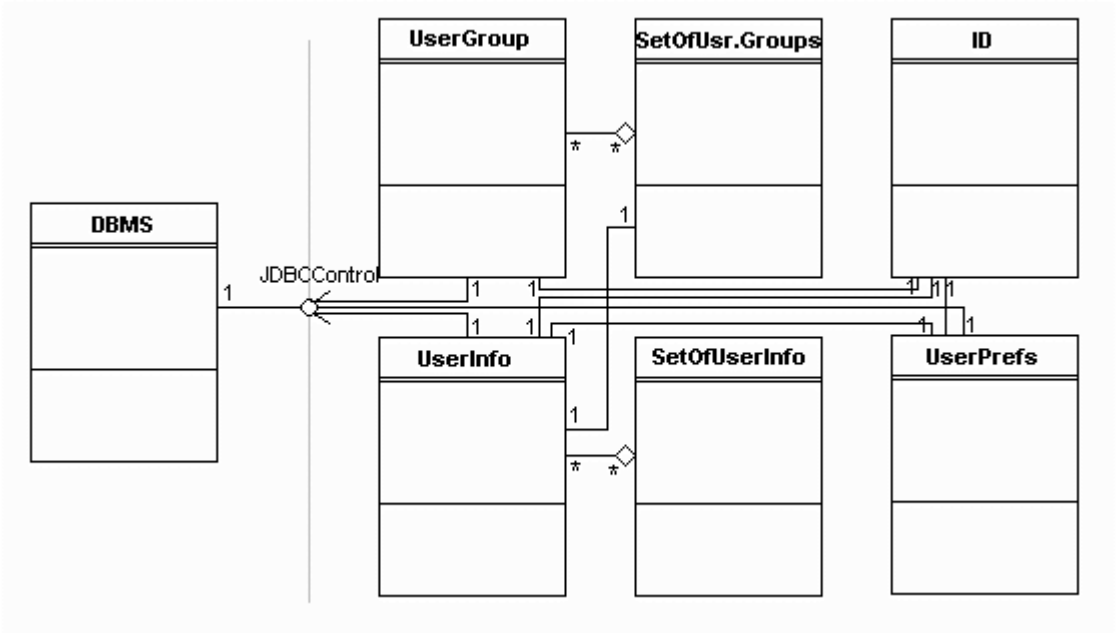


Figure 6: *Users and UserGroups*

5. Component descriptions

The ESA-standard ([ESA]) prescribes a structure for component descriptions.
Each component is identified by a name. The first paragraph specifies what type of component we deal with. The software requirements implemented by the component are described in the second paragraph. In the third paragraph there is a functional description. In the fourth paragraph the subordinates are described. In this paragraph we will present the classes contained in this component. The fifth paragraph describes which components are dependent on this component and on which components this component is dependent. In the sixth paragraph the interface is specified. Any non-trivial resources needed by the components are mentioned in the seventh paragraph. Any references for this component are mentioned in paragraph number eight. All attributes used are explained in the data paragraph. More information about this can be found in [ESA].
For an overview of the main system components see also figure 2.

5.1. Servlet

5.1.1. Type

The Servlet is a module that extends a request/response-oriented server.

5.1.2. Purpose

Enable the client to interact with the system via a web browser .
The Servlet component implements the following software requirements:
SR201, SR202, SR302, SR601

5.1.3. Function

Handle all incoming requests from clients and forward the requests to the Event Handler. The client uses a web browser to sent HttpRequests to the servlet. The Servlet handles the request by sending a HttpServletRequest to the EventHandler, along with a HttpServletResponse that can be used to sent results back to the corresponding browser.

5.1.4. Subordinates

The component Servlet has the component EventHandler as its subordinate.

5.1.5. Dependencies

This component forwards requests to the EventHandler. Before being able to receive requests from clients, the Servlet needs the EventHandler to be available.

5.1.6. Interfaces

A client uses a web browser to interact with the system. The Servlet receives two objects:
a ServletRequest which encapsulates the communication from the client to the server and a ServletResponse, which encapsulates the communication from the servlet back to the client.

The Servlet forwards the ServletRequest and ServletResponse to the Event Handler.

```
void doGet(HttpServletRequest request, HttpServletResponse response)
PRE:  “request is a HttpServletRequest sent by a browser, containing an
      attribute PageID”
POST: “Event Handler is given the responsibility to handle the request that
came in.”
```

The method and doPost are similar; doPost handles the HEAD methods. Head is just like GET, except that the server returns only the headers (including content length) not the body we write.

5.1.7. Resources

This component requires a valid instance of Event Handler to operate.

5.1.8. References

None.

5.1.9. Processing

The servlet uses the HttpRequest and HttpResponse (received from the browser) to create a HttpServletRequest and HttpServletResponse (for the exact contents see chapter 2.3.1) that is passed on a LoggedInUser.

5.1.10. Data

The Servlet contains no data structures. It does contain part of the error handling (for example when the Event Handler on which it depends does not initialise).

5.2. Event Handler

5.2.1. Type

This component is an event handler.

5.2.2. Purpose

The Event Handler component implements the following software requirements:
SR101, SR102

5.2.3. Function

The Event Handler grants permission for or denies new account requests and logins. The Event Handler keeps track of all LoggedInUsers and Sessions, and keeps a record regarding which session corresponds to which LoggedInUser. It utilises this record to pass requests from the Servlet to the corresponding LoggedInUser.

5.2.4. Subordinates

The component EventHandler has the component LoggedInUser as its subordinate.

5.2.5. Dependencies

The component EventHandler depends on the Servlet component.

5.2.6. Interfaces

The EventHandler gets two objects from the Servlet. These objects are then forwarded to the LoggedInUser that represents the user from which the request came. The objects are a HttpServletRequest, which encapsulates the request from the client to the server and a HttpServletResponse, which encapsulates the response from the server back to the client.

void processRequest (HttpServletRequest request, HttpServletResponse response)
PRE: "request is a HttpServletRequest sent by a browser containing an attribute PageID and response is the corresponding HttpServletResponse"
POST: "If the session that corresponds to a request does not exist, and the PageID is a
login: a new LoggedInUser is created if the login is valid(*) and the user is added to the set of tuples (LoggedInUser, sessionId)
newaccount: a new account is created if the account request is valid(*).
If the session that corresponds to a request does exist, the request is transferred to the LoggedInUser that has sent the request, by calling its method processPageRequest(request, response);"

(*) the condition 'a login/new account is valid' is more precisely defined in [URD] CAR004 and the [DDD].

5.2.7. Resources

This component requires a valid instance of LoggedInUser to operate, for users with other requests then a new login or new account request. It also needs a valid instance of the Servlet.

5.2.8. References

None.

5.2.9. Processing

The EventHandler processes new login requests and new account requests, by interpreting the HttpServletRequest and denying or granting this user request depending on the info in the HttpServletRequest. In case a login or new account request is denied (for example in case of wrong password or credit card number) the EventHandler sends a corresponding errormessage to the user from which the request originated. In case a login or new account request is succesful, the Event Handler keeps track of the user's session and creates a LoggedInUser (identified by the sessionID, the unique identifier given by the browser).

5.2.10. Data

The servlet contains a set of (LoggedInUser, SessionID) tuples to map the user sessions to the LoggedInUsers.

5.3. LoggedInUser

5.3.1. Type

User abstraction type

5.3.2. Purpose

The following software requirements are implemented by this component: SR115

5.3.3. Function

The LoggedInUser component is responsible for the processing of all requests that come from the user that corresponds to this LoggedInUser. These requests are passed by the EventHandler component, and delegated to the PageSelector component.

5.3.4. Subordinates

The component EventHandler has the component PageSelector as its subordinate.

5.3.5. Dependencies

This component depends on the EventHandler component.

5.3.6. Interfaces

The LoggedInUser receives a HttpServletRequest and a HttpServletResponse from the EventHandler component. It passes both of these on to the PageSelector component.

processPageRequest(HttpServletRequest request, HttpServletResponse response)
PRE: "request is a HttpServletRequest sent by a browser, containing an attribute PageID"
POST: "Pageselect is given the responsibility to perform any actions requested through the request object, and return a requested page through the response object: pageselect.selectPage(request, response);
the info object is updated afterwards by using the getInfo() method."

String getName()
PRE: true
POST: returns the name of the LoggedInUser (using the userinfo object).

5.3.7. Resources

This component requires a valid instance of PageSelector and EventHandler to operate.

5.3.8. References

None.

5.3.9. Processing

At a new login, the constructor of LoggedInUser verifies if the login/password is correct.

5.3.10. Data

LoggedInUser contains an object UserInfo, containing the information about the user that corresponds to the LoggedInUser.

5.4. PageSelector

5.4.1. Type

PageSelector.

5.4.2. Purpose

The PageSelector component does not directly implement any particular software requirements.

5.4.3. Function

This component processes the actions from LoggedInUser by interpreting the HttpServletRequest, delegating update actions to the FormHandler component and page requests to the HTML Generator component.

5.4.4. Subordinates

The component PageSelector has the components FormHandler, HTML Generator and DataClasses as its subordinates.

5.4.5. Dependencies

This component depends on the LoggedInUser component.

5.4.6. Interfaces

```
void selectPage(HttpServletRequest request, HttpServletResponse response)
PRE:  “request contains a valid pageID attribute”
POST: “FormHandler is given the responsibility to perform any required
      update, HTML Generator is given the responsibility to return the
      correct HTML-page to the user”
```

5.4.7. Resources

The PageSelector component does not require any resources.

5.4.8. References

None.

5.4.9. Processing

This component retrieves and interprets the pageID attribute from the HttpServletRequest, then depending on its interpretation of the pageID selects the right HTML-page to be generated by the HTML Generator component, and if necessary selects the right FormHandler to handle a required update.

5.4.10. Data

PageSelector contains no data structures.

5.5. HTML Generator

5.5.1. Type

HTML Generator.

5.5.2. Purpose

The following software requirements are implemented by this component:
SR201, SR203, SR501 and partly:
SR107, SR108, SR111, SR116, SR118, SR121, SR122, SR127, SR128.

5.5.3. Function

This component generates HTML-pages and returns them to the user through the HttpServletResponse object.

5.5.4. Subordinates

The component HTML Generator has the component DataClasses as its subordinates.

5.5.5. Dependencies

The component HTML Generator depends on the PageSelector component.

5.5.6. Interfaces

For each HTML Generator page, a separate class is defined. Every class in HTML Generator has the following interface:

```
generatePage(request, response, user);
PRE:  response = RESPONSE && user = USER &&
      “REQUEST contains the necessary attributes for this page”
POST: “this page is generated according to the attributes in RETURN and
      returned to USER through the respone object”
```

Each page-class uses its own descriptive name, such as generatePageBar, generatePageIndex, generatePageSideBar and generatePageMain. The PageSelector attribute is responsible for selecting the correct page class.

5.5.7. Resources

The HTML Generator component does not require any resources.

5.5.8. References

None.

5.5.9. Processing

The HTML Generator processes the HttpServletRequest, retrieving relevant data to construct the requested page.

5.5.10. Data

HTML Generator contains no data structures.

5.6. DataClasses

5.6.1. Type

This component consists of a set data classes that represent the data stored in the store.

5.6.2. Purpose

The following software requirements are implemented by this component:
SR105 - SR129

5.6.3. Function

The DataClasses represent the object layer for the store, i.e. for each type of stored element (folder, metadocument etc.) a class is defined which handles create, read, update and delete actions on the type. Also for some types sets have been defined (such as SetOfFolders and SetOfMetaDocuments).

5.6.4. Subordinates

The component DataClasses has the component JDBCControl and the folder structure of the store as its subordinate.

5.6.5. Dependencies

This component depends on the Formhandler, HTML Generator and PageSelector components.

5.6.6. Interfaces

Each Dataclass also has set- and get-methods defined on their attributes, of the following form (where ‘attribute’ represents an attribute of a Dataclass ‘Dataclass’):

```
void setAttribute(AttributeType newvalue);
PRE:  newvalue = NEWVALUE
POST: Dataclass.attribute = NEWVALUE
```

This method implements the ‘update’ functionality on storables.

```
AttributeType getAttribute();
PRE:  TRUE
RET:  Dataclass.attribute
```

This method implements the ‘read’ functionality on storables.

Furthermore, multiple constructors are defined to distinguish whether the data class represents an existing storable or a new storable. These methods implement the ‘create’ functionality on storables (for non-existant

```
void delete();
PRE:  TRUE
RET:  “this item is removed from the database”
```

Note that these interfaces do not apply to the ID data class – its interface is managed by the JDBC Control component (see 5.7: JDBC Control).

5.6.7. Resources

The DataClasses component does not require any resources.

5.6.8. References

None.

5.6.9. Processing

The DataClasses component translates updates and requests into SQL, and retrieves data from the ResultSets it receives from its subordinate component JDBCControl.

5.6.10. Data

Each data class has a set of attributes, in which are stored the attributes of the storable represented by the data class.

5.7. JDBC Control

5.7.1. Type

The component JDBC Control is an interface class.

5.7.2. Purpose

There are no particular software requirements implemented by this component.

5.7.3. Function

The JDBC Control is used to provide and standardise the interface to the DBMS, thus increasing the modularity of the connection with the database.

5.7.4. Subordinates

This component has the Database component as its subordinate.

5.7.5. Dependencies

This component depends on the HTML Generator, the FormHandler and the PageSelector components.

5.7.6. Interfaces

```
void InitDB(string db_url);
PRE:  “db_url is the location of the DBMS.”
POST: “A connection with the database via the JDBC-driver is established”

public static void closeDB();
PRE:  “A connection with the database via the JDBC-driver has been
      established”
POST: “The connection with the database has been terminated”

ResultSet queryDB(string query);
PRE:  query = QUERY &&
      “QUERY represents a valid query in the sql-syntax”
RET:  “a set of items containing the rows of the response from the DMBS to
      the query QUERY”
```

```
void commandDB(string command);
PRE:  command = COMMAND
POST: “The DBMS has executed COMMAND”

ID pickID();
PRE:  TRUE
RET:  “an ID object representing a unique, unused identification number”
```

5.7.7. Resources

The JDBC Control component requires an SQL-database and the JDBC driver for that database.

5.7.8. References

[MySQL], [JDBC]

5.7.9. Processing

JDBC Control does not process any data – the underlying DBMS processes the data sent by JDBC Control.

5.7.10. Data

JDBC Control contains one data structure: a “Connection”. This data structure provides an interface to the JDBC-driver and is bundled with the driver.

5.8. Form Handler

5.8.1. Type

Update action handler component

5.8.2. Purpose

The following software requirements are partly implemented by this component:
SR109, SR110, SR112-SR114, SR117, SR119, SR120.

5.8.3. Function

The FormHandler component is responsible for performing all updates on the databases. It receives update requests and translates these into updates on various storables, which are then handled by the DataClasses component. The FormHandler component consists of a set of classes, each handling a specific update action.

5.8.4. Subordinates

The Formhandler component has the DataClasses component as its subordinate.

5.8.5. Dependencies

The Formhandler component depends on the PageSelector to select the required FormHandler class.

5.8.6. Interfaces

The FormHandler component consists of several classes which generally have the following interfaces (where ‘Purpose’ indicates which update the specific class implements):

```
void HandleFormPurpose(HttpServletRequest request, UserInfo user);
```

```
PRE:  “request contains the necessary information for the update to
      be performed”
POST: “the update of this class has been performed”
```

Some FormHandler classes also require a metadocument:

```
void HandleFormPurpose(HttpServletRequest request, UserInfo user,
                  MetaDocumentmetadoc);
PRE:  “request contains the necessary information for the update to
      be performed”
POST: “the update of this class has been performed”
```

5.8.7. Resources

None.

5.8.8. References

None.

5.8.9. Processing

The FormHandler component interprets the HttpServletRequest to retrieve required data to perform the upload.

5.8.10. Data

FormHandler has no datastructures.

5.9. Database Component

5.9.1. Type

The component Database is a database.

5.9.2. Purpose

The following software requirements are implemented by this component:
SR105 – SR129, SR301, SR302, SR502

5.9.3. Function

In the database metadocuments and similar data are stored, updated, retrieved and deleted. It serves as a permanent data structure, handled by a DBMS (DataBase Management System).

5.9.4. Subordinates

The subordinates of the Database component are the following tables:
ACCESSRIGHTS
CLASSIFICATION
DIGITALREPRESENTATION
FOLDER
ID
METADOCUMENT
METAINFORMATION
METATYPE
NEXTSTATUS
PHYSICALREPRESENTATION
REFERENCE

STATUS
USERGROUP
USERGROUPCLASSIFICATION
USERGROUPUSERINFO
USERINFO
USERPREFS

5.9.5. Dependencies

Because it is an external component, the Database component does not depend on any other component.

5.9.6. Interfaces

The interfaces of the Database component consist of the interfaces offered by the DBMS. This consists of a DBMS server, to which a connection can be made using the JDBC (Java DataBase Control) software. The relevant interface receives SQL queries and returns a standard data object containing the results (ResultSet, see [JDBC]).

5.9.7. Resources

This component requires enough memory and hard disk space to accommodate the stored information, and also runs on a system that complies with the minimum system requirements as demanded by [MySQL].

5.9.8. References

[MySQL], [JDBC]

5.9.9. Processing

The processing of the Database component is handled by the DBMS, and is therefore not part of our system.

5.9.10. Data

Internal data of the Database consists of the database tables. See the table descriptions below.

5.10. FolderStructure

5.10.1. Type

Native storage structure.

5.10.2. Purpose

The following requirements are (partially) implemented by the FolderStructure component:
SR105, SR106, SR108, SR109, SR110,
SR112, SR113, SR114, SR116, SR118,
SR119

5.10.3. Function

FolderStructure contains native representations of digital representations of metadocuments. Storing these digital representations directly into the database would yield a solution that does not comply with the user requirements ([URD] CAR001) therefore the need for FolderStructure arose.

FolderStructure acts as a representation depository; communication with the component is handled by the operating system of the store computer.

5.10.4. Subordinates

The FolderStructure component has no subordinates.

5.10.5. Dependencies

FolderStructure does not depend on other components: it requires the operating system of the store computer to be running.

5.10.6. Interfaces

All interfaces with the FolderStructure component are handled by the operating system of the store computer.
FolderStructure has no interfaces with other components.

5.10.7. Resources

FolderStructure requires sufficient hard disk space on the store computer to contain all digital representations stored in the system, as well as the folder structure.

5.10.8. References

None.

5.10.9. Processing

There are no processing elements in the FolderStructure component.

5.10.10. Data

The data of the FolderStructure component consists of the digital representations of metadocuments, as well as the folder structure in which these representations are stored.

6. Feasibility and resource estimates

This section gives a summary of the computer resources required to build, operate and maintain the software.

6.1. Resources to build the system

The following resources are needed to build the eArchive system:

- 4 Intel Pentium II 400 MHz MMX processor with 128 MB primary and 8 GB secondary store, running as operating system Microsoft Windows NT (or compatible hardware, with at least the same kind of functionality and performance);
- 1 Intel Pentium II 500 MHz MMX processor with 128 MB primary and 8 GB secondary store, running as operating system RedHat Linux 6.1, which is used as FTP server for both code and documentation storage (or compatible hardware, with at least the same kind of functionality and performance).

6.2. Resources to operate the system

The following resources are needed to operate the eArchive system:

when the system is configured for internal use:

- as server a computer with a Pentium II 233 MHz processor and 128 MB primary and 8.4 GB secondary store, running as operating system RedHat Linux 6.1 (or compatible hardware, with at least the same kind of functionality and performance). The server has a continuous Intranet connection
- as client a computer with a 1 Mbit connection to the company Intranet. With a browser from Netscape (at least version 4.6) or Microsoft (at least version 4).

when the system is configured for external use:

- as server a computer with a Pentium II 233 Mhz processor and 128 MB primary and enough secondary store to accommodate all user request, running as operating system RedHat Linux 6.1 (or compatible hardware, with at least the same kind of functionality and performance). The server has a continuous Internet connection
- as client a computer with a 1Mbit connection to the Internet, with a browser from Netscape (at least version 4.6) or Microsoft (at least version 4).
-

6.3. Resources to maintain the system

There are no resources needed to maintain the eArchive system because there is no maintenance on the software after the TR phase involved with the project.

7. Requirements traceability matrix

1240

AD Components to SR Matrix

Component	Software Requirement
Servlet	SR201, SR202, SR302, SR601
Event Handler	SR101, SR102
LoggedInUser	SR115
PageSelector	none directly
HTML Generator	SR201, SR203, SR501 Partly*: SR107, SR108, SR111, SR116, SR118, SR121, SR122, SR127, SR128
DataClasses	SR105 – SR129 A number of these requirements coincide with objects in the store. Those requirements are listed both here and at the folder structure/database
Form Handler	Partly*: SR109, SR110, SR112-SR114, SR117, SR119, SR120
JDBC Control	no particular requirement
Database	SR105 – SR129 SR301, SR302, SR502
FolderStructure	SR105, SR106, SR108, SR109, SR110 SR112, SR113, SR114, SR116, SR118, SR119
General	SR303, SR304, SR401, SR501
	* these requirements involve operations on some of the dataclasses. Since this component is the one that enable use of these operations, the requirements are listed both here and at the dataclasses

SR Components to AR Matrix

Component	Software Requirement
SR101	Event Handler
SR102	Event Handler
SR105	DataClasses, Database, FolderStructure
SR106	DataClasses, Database, FolderStructure
SR107	DataClasses, Database, (HTMLG)
SR108	DataClasses, Database, FolderStructure, (HTMLG)
SR109	DataClasses, Database, FolderStructure, (Form Handler)
SR110	DataClasses, Database, FolderStructure, (Form Handler)
SR111	DataClasses, Database, (HTMLG)
SR112	DataClasses, Database, FolderStructure, (Form Handler)
SR113	DataClasses, Database, FolderStructure, (Form Handler)
SR114	DataClasses, Database, FolderStructure, (Form Handler)
SR115	LoggedInUser, DataClasses, Database
SR116	DataClasses, Database, FolderStructure, (HTMLG)
SR117	DataClasses, Database, FolderStructure, (Form Handler)
SR118	DataClasses, Database, FolderStructure, (HTMLG)
SR119	DataClasses, Database, FolderStructure, (Form Handler)
SR120	DataClasses, Database, (Form Handler)
SR121	DataClasses, Database, (HTMLG)
SR122	DataClasses, Database, (HTMLG)
SR123	DataClasses, Database
SR124	DataClasses, Database
SR125	DataClasses, Database
SR126	DataClasses, Database
SR127	DataClasses, Database, (HTMLG)
SR128	DataClasses, Database, (HTMLG)
SR129	DataClasses, Database
SR201	Servlet, HTMLG
SR202	
SR203	HTMLG
SR301	Database
SR302	Servlet, Database
SR303	All/General
SR304	All/General
SR401	All/General
SR501	All/General
SR502	Database
SR601	Servlet

1245 (): partly implemented by (see AD to SR matrix).