Investigation and development of a home banking site

Kim Hansen – C938223

Informatics and Mathematical Modelling
Technical University of Denmark

17. April 2002

Preface

This report represents the final project of the civil engineering course at the Technical University of Denmark.

The report was done in EF tecnologias in Portugal in cooperation with the Institute for Informatics and Mathematical Modelling at the Technical University of Denmark in the period from 17.10.2001 until 17.04.2002.

Kim Hansen

Abstract

In this thesis a home banking application is investigated along with the security issues surrounding it. The home banking application has to be able to communicate with a real bank in an efficient way, where the bank system is clearly separated from the home banking application.

A model for developing automated web site applications is presented. This model separates the concerns in web site design and provides the basis for a generation mechanism for hypermedia design.

A model to ensure a secure environment for the home banking application is implemented.

Table of Contents

1. INTRODUCTION 8
1.1 Motivation and background 8
1.2 Description of a home banking site 9
1.3 Example System of the home banking site 9
1.4 Goal and limitations of the case study 9
1.5 Explanation of the terms and definitions used in this report 10
1.6 Requirements to the reader 10
1.7 Overview of the thesis 11
2. SYSTEM ARCHITECTURE 12
2.1 Dynamically versus statically generated sites 12
2.2 Connection between home banking application and bank legacy system 13
2.3 Web Services and how they can be used for home banking sites 13
2.4 System Architecture Diagram 14
3. SECURITY ASPECTS 16
3.1 Main security considerations in a home banking site 16
3.2 Security Model for the home banking application 17
4. HOME BANKING DESIGN MODEL 22
4.1 Modelling a Web Site 22
4.2 Requirements Specification 22
4.3 Functionality Description - Conceptual Model 23
4.3.1 Definition of Functions 23
4.3.2 Access Specification 24
4.3.3 Logging Specification 24
4.3.4 Database Modelling 25
4.4 Navigational Design 29
4.4.1 Navigation Space Model 29
4.4.2 Navigational Structure Model 30
4.5 Layout Design 31
4.5.1 Presentational Model 32
4.5.3 Applying CSS – Style, Colours, Fonts, Images 33
5. IMPLEMENTATION 36
5.1 Choice of technology – Java technology (JSP and Servlets) 36
5.2 MVC - Model View Controller (Java Server Pages Model 2) 38
5.3 Implementation Structure 38
5.4 Mapping navigation and functionality 44
5.5 Mapping views 46
5.6 Adding CSS 47
5.7 Implementing the Authentication Model and HTTPS 49
5.8 Bank Simulator – Glue Web Service 51
5.9 Problems and considerations during implementation 55
7. DOCUMENTATION OF A HOME BANKING SITE 56
7.1 Documentation of each web page 56
7.2 Web Site Map 57

8. FUTURE EXPANSIONS 58
8.1 Improving Application Functionality 58
8.2 Administration Application 58
8.3 Expanding model to support multi-channels using the Web Service format 58
8.4 Improving separation at top level in the model 59
8.5 Improving security in the implementation 59
8.5 Implementing PKI-security 60

9. CONCLUSION 62
9.1 Modelling a web site 62
9.2 Security in a home banking application 63
9.3 Web Service as a communication platform 63
9.4 Documentation of a home banking application 63
9.5 MVC as implementation platform 64

APPENDIX .... ERROR! BOOKMARK NOT DEFINED.

A. SUMMARIES AND COMMENTS ON RELEVANT ARTICLESERROR! BOOKMARK NOT DEFINED.
Web Service Technology .... **Error! Bookmark not defined.**
OOHDM – Object Oriented Hypermedia Design Model – short overview .... **Error! Bookmark not defined.**
Middleware Challenges Ahead .... **Error! Bookmark not defined.**
More about Web Security .... **Error! Bookmark not defined.**
XML’s Impact on Databases and Data Sharing .... **Error! Bookmark not defined.**
Utilizing Abstract WebEngineering Concepts: an Architecture .... **Error! Bookmark not defined.**
WSDL file describing the bank operations as a web service: .... **Error! Bookmark not defined.**

B. SOURCE CODE .... ERROR! BOOKMARK NOT DEFINED.
Home Banking Application .... **Error! Bookmark not defined.**
ConsultationAction.java .... **Error! Bookmark not defined.**
OrderChequesBean.java .... **Error! Bookmark not defined.**
XMLConfigLoader.java .... **Error! Bookmark not defined.**
ControllerHook.java .... **Error! Bookmark not defined.**
BankingHook.java .... **Error! Bookmark not defined.**
Populate.java .... **Error! Bookmark not defined.**
BeanUtils.java .... **Error! Bookmark not defined.**
GenericBean.java .... **Error! Bookmark not defined.**
Resource.java .... **Error! Bookmark not defined.**
GenericAction.java .... **Error! Bookmark not defined.**
Beans.java .... **Error! Bookmark not defined.**
Auth.java .... **Error! Bookmark not defined.**
Action.java .... **Error! Bookmark not defined.**
ActionContext.java .... **Error! Bookmark not defined.**
HomeBankOperations.java .... **Error! Bookmark not defined.**
IHomeBankOperations.java .... **Error! Bookmark not defined.**
HTMLElements.java .... **Error! Bookmark not defined.**
Global.java .... **Error! Bookmark not defined.**
DBConnect.java .... **Error! Bookmark not defined.**
Config.java .... **Error! Bookmark not defined.**
Bank Simulator Application .... **Error! Bookmark not defined.**
Pub.java .... **Error! Bookmark not defined.**
BankDatabase.java .... **Error! Bookmark not defined.**

BankOperations.java ..... **Error! Bookmark not defined.**
IBankOperations.java ..... **Error! Bookmark not defined.**
styles.css ..... **Error! Bookmark not defined.**
balance.jsp ..... **Error! Bookmark not defined.**
transactionList.jsp ..... **Error! Bookmark not defined.**
stockList.jsp ..... **Error! Bookmark not defined.**
transfer.jsp ..... **Error! Bookmark not defined.**
payment.jsp ..... **Error! Bookmark not defined.**
orderCheques.jsp ..... **Error! Bookmark not defined.**
newName.jsp ..... **Error! Bookmark not defined.**
loadBean.jsp ..... **Error! Bookmark not defined.**
menu.jsp ..... **Error! Bookmark not defined.**
welcome.jsp ..... **Error! Bookmark not defined.**
errorPage.jsp ..... **Error! Bookmark not defined.**
errorLogin.jsp ..... **Error! Bookmark not defined.**
header.jsp ..... **Error! Bookmark not defined.**
index.jsp ..... **Error! Bookmark not defined.**
loginLevel1.jsp ..... **Error! Bookmark not defined.**
loginLevel2.jsp ..... **Error! Bookmark not defined.**
loginLevel3.jsp ..... **Error! Bookmark not defined.**
loginInclude.jsp ..... **Error! Bookmark not defined.**
Configuration files ..... **Error! Bookmark not defined.**
auth-config.xml ..... **Error! Bookmark not defined.**
action-config.xml ..... **Error! Bookmark not defined.**
web.xml ..... **Error! Bookmark not defined.**

1. Introduction

The main purpose of this project is to investigate the aspects involved in web site creation. It is based on a case study of a home banking site where fast easy navigation, logic presentation of information and security aspects have high priority. These aspects cover several problems like efficient web site creation methods, effective communication with a bank and the security concerns surrounding it.

1.1 Motivation and background

With home banking sites and web sites in general becoming increasingly bigger and more complex, there is a growing need for structured implementation methods.

Where content is changed on the fly,
New services are constantly added,
New navigation and interface features added,
Easy connection to different legacy systems that change and get updated

There doesn't exist one perfect solution to deal effectively with these problem, but investigation around this area is popularly called "web engineering". Web engineering tries to define models to automate the process of generating a web site and separate the main concerns.

This project is partly motivated by the report: "A method to develop web-based systems" by Niels Bach[3] where an automated method of implementing a web site is developed. This method uses a functional approach built around a web site with a database. In this project a home banking site is investigated and with it also the surrounding elements. It also uses a database, but the functionality is not only defined by the application itself but also provided by a bank. This brings other problem considerations into the picture. How does the interaction with the bank work? Could it benefit from the new concept of *web services*?

Looking at a web site it can be split up into three main parts: layout, interactivity/dialog and functionality. Each of these parts has to be fully integrated, but play a different role on the web site, have different requirements and may very likely be implemented by different people. By splitting it up, different people with different skills can specialize in their part of the site.

The three main parts that constitute a web site are:

Layout – the layout of a site is the graphical presentation of the site, with placement of text, pictures, forms, tables etc.

Interactivity/dialog – the interactivity and user-dialog deals with the user interaction like menu navigation, site-maps etc.

Functionality – the functionality of the site defines the calculations and operations that is done in the background. This is where the business logic is implemented.

One of the goals is to achieve a model that efficiently supports reuse mechanisms. Aiding both the design process and the maintenance of a web site.

1.2 Description of a home banking site

A home banking site is normally a small part of a complete bank site. The complete bank site contains a great deal of information and functionality and the target group is everybody from individual clients, company clients, partners, to the general public. The home banking part is targeting individual clients with accounts in the bank. The Internet service is just one way of accessing the home banking site. Other ways include WAP (Wireless Application Protocol), WebTV and telephone service. In this case study the focus is on accessing the home banking site through the Internet with a web browser. The home banking site is a service restricted only to clients of the bank.

A home banking site provides the client of the bank with the possibility of doing banking operations with a web browser connected to the Internet. Generally the user has the possibility of consulting the balance, consulting the bank transactions, order cheques, execute money transfers, payments, stocks and funds operations, site personalization, organize messages among other things.

1.3 Example System of the home banking site

The example system is a part of the whole home banking site. It is used to put a focus on the important aspects and problems of the design and realization of the site, rather than showing the implementation of all the functionalities.

Informal description:
For the example system the informal description could be something like:

At first a welcome page is presented to the user. The user is a client of the bank and needs to identify himself to access the home banking application. His next step will either be to press the Login button in the menu or choose a functionality. In either case he will be presented with a login authentication page. If the client id and password are accepted, a main page with client information is shown along with a menu with navigation. From here, the user can choose to go to a consult page with the possibility of going to a balance page to check the balance for each account or go to a bank operation list page to see a list of the latest bank operations done. Another possibility from the main page is to go to an operations page. In this page the client can access a transfer page to transfer money from one account to another or go to a payment page where payments can be done. The last option on the main page is to go to a personalize page where the client can set the names for his accounts.

For the example system the following functionality has been chosen:

Consult Balance – shows the balance for a given account
Consult Transactions – shows a list of transactions for a given account
Execute Transfer – a money transfer from one account to another
Execute Payment – a service payment
Rename Account – applying a personal name for an account

1.4 Goal and limitations of the case study

A home banking site involves a great deal of aspects, covering everything from layout, marketing, security, functionality, bank communication, data storage, etc. In a real life scenario several teams work in collaboration to implement the site, each specializing in their part. The goal of this case study is to get around all the main concerns and to develop a

method that helps the separation of the different tasks involved and the automation of the whole process.
Some limitations to the case study are needed since several of the mentioned aspects are huge. To go into complete detail with every aspect is not possible with the given time period of the project. For instance, security itself is an ever expanding area with new and improved ways appearing on a daily basis. For this project the security issues involved with a home banking application are investigated and a model for setting up a secure environment is described.
Another rapidly evolving subject is *web services*. In this project it is only touched superficially. The main advantages are explained and an example is given.
To administrate a home banking site it is necessary to have an administration application that can add, delete users, administrate accounts etc. This application would use the same database as the home banking application, but other than that is a separated application. It is outside the scope of this case study to build an administration application.

1.5 Explanation of the terms and definitions used in this report

The World Wide Web (WWW) is most often called **the Web**.
The Web is a network of computers **all over the world**.
All the computers in the Web can **communicate with each other**.
All the computers use a **communication standard called HTTP**.

Web information is stored in documents called **Web pages**.
A collection of Web pages is called a **Web site.**
Web pages are files stored on computers called **Web servers**.
Computers reading the Web pages are called **Web clients**.
Web clients view the pages with a program called a **Web browser**.
Popular browsers are **Internet Explorer and Netscape Navigator**.

A browser fetches a Web page from a server by a **request**.
A request is a standard HTTP request containing a **page address**.
A page address looks like this: http://www.someone.com/page.htm.

1.6 Requirements to the reader

It is assumed that the reader has extended knowledge about Java, HTML and the usage of the Internet. Furthermore the section about security assumes some knowledge about general Internet technology.

1.7 Overview of the thesis

In the following an overview of the thesis is shown with the main points described:

Introduction – The introduction to the thesis along with a description of a home banking site and the example system used throughout the report

System Architecture – Describes the system architecture and the components needed to implement a home banking application

Security Aspects – Describes the security aspects involved and builds a security model that ensures a secure environment

Home Banking Design Model – A model for the home application is built and every step from the requirements specification to the concepts used in the implementation is shown

Implementation – Describes the chosen technology and the implementation model used along with the mapping from the home banking design model to the implementation framework

Comparison with other models – Compares the model introduced in the thesis with other existing models

Documentation – Describes a way to document a home banking application

Future Expansions – Gives suggestions to future work and improvements

Conclusion – Summarizes what has been developed and the experiences learned

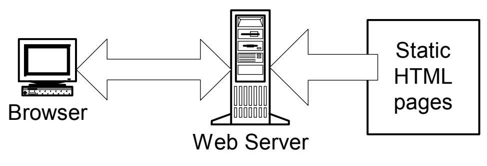
Appendix – Contains program code along with resumes of relevant articles used in the thesis and some configuration and installation notes

2. System Architecture

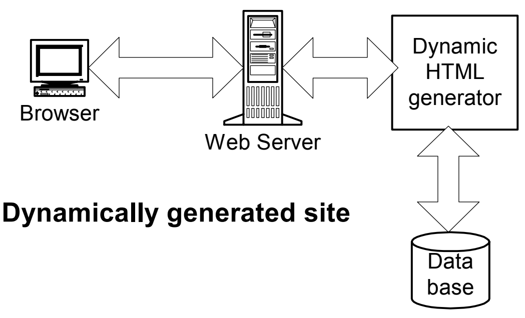
In this chapter the system architecture for the home banking application is investigated. Before the actual system architecture diagram is shown, the concepts and technologies involved are described.
At first a dynamically generated site is compared with a statically generated site. This is followed by the technology needed to implement dynamically generated sites. Then the connection between the home banking application and the bank system is described and the web service format is introduced as a practical solution for the communication. In the end the system architecture diagram is shown and explained.

2.1 Dynamically versus statically generated sites

Deciding whether a site should be implemented as a statically site or a dynamically generated site is determined by the underlying functionality. A site where the pages exists as files is a static site, whereas a site where the pages are results of computations triggered every time a document is requested has to be implemented as a dynamically generated site. Most of the content on the Internet is static and doesn't change or only change very infrequently, in spite of being generated from templates and database systems. In these cases the entire content of a site may be "compiled" before application deployment and static pages be stored as files directly accessible by the server (see figure below). An example of this could be a presentation site, e.g. for a new movie. Once the presentation site has been made it typically doesn't change in the time it is online. In other cases where the content changes frequently (e.g. a news site) and/or depends on user input (e.g. a home banking site) this approach is not feasible. Even in these cases it is a good idea to split up the site in sub-sites, where the static part is one sub-site and the dynamic part is another sub-site.
A home banking site is a good example of a site where the pages need to be created on the fly according to requests from the user. The content to be displayed is either information that changes very frequently e.g. the balance of an account or depends on the input data given e.g. the result of a money transfer. Typically the user provides some input e.g. the account number and receives a dynamically generated page, in this case with the balance possibly along with other information for that account. In the figure below is shown the main difference between a statically generated site and a dynamically generated site.



Statically generated site



Dynamically generated site

The platform for providing dynamic web sites with database access consists of a web server and a database server. The web server runs a web application that accesses the database server.

Web Server
A web server is a program that provides network access to web pages. Its main job is to respond to HTTP requests from web clients. A request is in the form of a URL (Uniform Resource Locator). When a request is made the web server typically it checks if the file exists and is accessible and then returns the page to the client. Every computer on the Internet that contains a web site must have a web server program to be able to respond to requests. In this project the iPlanet web server[13] from Netscape is used.

Web Application
The web application is the collection of files that the web server uses during runtime. In this project the web application includes servlets, JavaServer Pages, HTML documents, and other web resources which might include image files, style sheet files, compressed archives, and other data.

Database Server
A database server is a program running a database and responding to SQL requests. The database keeps the data for the application in an organized way, so it is easily accessed, managed and updated.
Most program languages have a database interface used to communicate with the database. The database runs individually from other programs and only needs the interface for the program to access it. In this project the MySQL database[8] and JDBC[24] interface are used.

2.2 Connection between home banking application and bank legacy system

The home banking application can be seen as an interface program between the real bank and the client. When an operation is done in the home banking site it has to be reflected in the bank itself. All the banking operations are done inside the bank with the bank legacy system. The home banking application provides the channel for accessing part of the functionality in the bank through a web browser. It is mainly a service given by the bank to the clients. The home banking application has to organize which operations are possible through the browser. Some functionality is better suited for going to the bank (e.g. if personal contact is required) while other functionality is convenient to do from home.

2.3 Web Services and how they can be used for home banking sites

One of the newest technologies and the buzz-word of the day is Web Services. Basically a Web Service is a web-based application that can dynamically interact with other web applications using well defined and well spread standards. The Web Service format is build around using standards like XML (eXtendable Markup Language)[18], SOAP (Simple Object Access Protocol)[15] and WSDL (Web Service Description Language)[16]. The power of XML lies in the way it can describe the format of data to be interchanged in a standard way. It means that two or more parts that need to interchange data doesn’t have to know each others system and data format. By agreeing on a format defined in XML they can concentrate on their own system. This means that changing, updating etc. one of the parts doesn’t interfere with any other part.
The other standards used in the web service format and in the home banking application are SOAP and WSDL.
SOAP is a XML based protocol. It provides a way to access services, objects and servers in a completely platform-independent way. With SOAP it is possible to query, invoke,

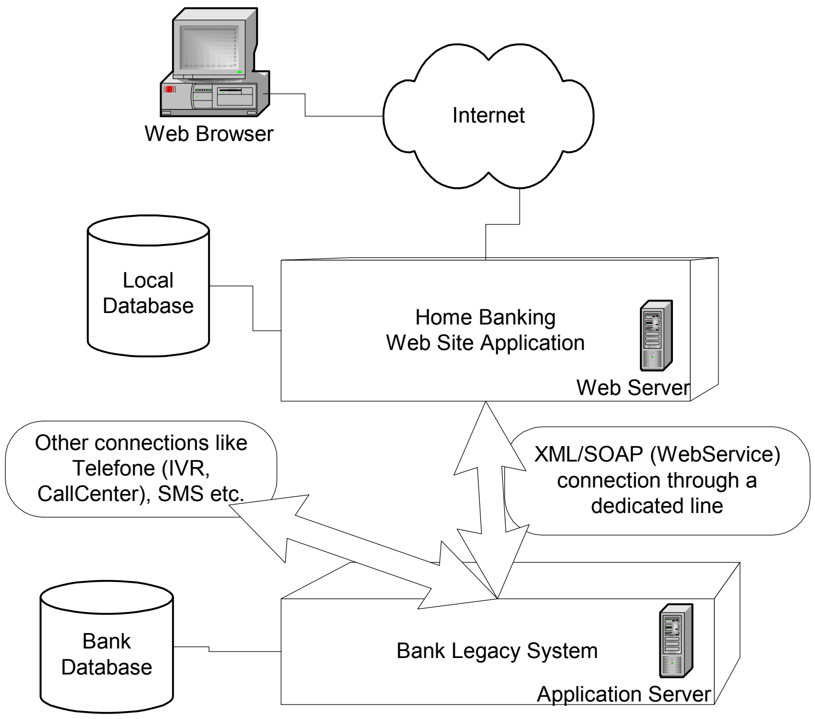
communicate with services provided on remote systems, without regard to the remote systems location, operating system or platform.
WSDL is an XML based language used to define a web service and describe how to access it. Normally it is not necessary to understand WSDL to use it because there are tools that automatically generate the WSDL file. In chapter 5.7 an example of an automatically generated WSDL file for the bank web service is given.
A home banking site is a service provided by a real bank. It is an application extension to the already existing bank system. This means that the home banking application needs to communicate with the bank itself for each bank operation done by the client. The more independent in terms of platform and implementation the two systems are the better for obvious maintenance reasons. Furthermore the bank legacy system has typically existed a long time before the home banking application. By agreeing on a standard way of communication like SOAP and using a format optimised for describing data to be interchanged like XML, the home banking application is effectively separated from the bank application.
Adding to the separation of the systems the web service format was chosen for the way it can easily support an object-oriented design model which is the base for the model described in this thesis. Internally the web services are implemented with object-oriented languages following an object-oriented design model. Externally the web services appear to be objects accessible by standard interface-descriptions. In this project the communication between the bank and the home banking application is done using the web service format to directly invoke bank operations using SOAP and XML. Both the home banking application and the bank implementation is based on an object-oriented platform.

2.4 System Architecture Diagram

In the figure below is shown the system architecture of the home banking application. It is accessed through the Internet with a web browser (Netscape, Internet Explorer etc.)

In the bottom level of the system architecture we find the bank legacy system which is the core of the bank. Inside the bank legacy system is where all the possible banking operations are defined. This system has the main database with all the information about accounts, clients, etc.. Anybody or anything that wants to access this system has to connect either through a terminal system with authentication or some other application also implementing authentication.
Not shown in the diagram is the machine and software in between the bank legacy system and the basic component structure which provides the information as a web service (using XML/SOAP). It is assumed in this case study that this software exists (which normally is also the case in real banks) and the connection to the bank functions as a web service.

The next level is the home banking application where the main functionality is defined as well as security services. At this level we also have a database where user, account and error information related to the home banking application is defined. Furthermore a part of the functionality at this level is the logging of all user interaction with the home banking site.


Web Browser
Internet
Local Database
Home Banking Web Site Application
Web Server
Other connections like Telefone (IVR, CallCenter), SMS etc.
XML/SOAP (WebService) connection through a dedicated line
Bank Database
Bank Legacy System
Application Server

System Architecture Diagram

The line between the web server hosting the home banking application and the application server hosting the bank legacy system is a dedicated line. That it is dedicated means that the communication is done on a physical connection between the two systems unshared with anyone else.

Since there is no real bank available to communicate with for this project I have implemented a bank simulator. The bank simulator provides the bank operations as a web service. In chapter 5.7 it is described how the bank simulator was implemented.

3. Security Aspects

In this chapter I will first describe the security threats to be considered in a home banking site and the security services needed to ensure a secure environment. In the second part a model is built that implements each of the necessary security services for the home banking application.

3.1 Main security considerations in a home banking site

Computer security and in particular web security is becoming increasingly bigger and more important as the Internet expands and new online business technology evolves. There is a growing need for secure web services. A home banking site is a good example of a web service where security plays an extremely important role. Both the bank and the client want full privacy from unauthorized parties.

To understand what web security involves it is split up into several aspects[2]. Each of these aspects can be seen as a security service enhancing the systems capability of avoiding a security attack.

- **Authentication:** To ensure the identity of the involved entities
- **Access Control:** To protect the system resources against unauthorized access according to a security policy
- **Audit Trail:** A recording of the activities of an entity, sufficient enough to recreate the activities at a later stage
- **Confidentiality:** Ensuring that the system resources are only available to authorized entities
- **Integrity:** To ensure that the information has not been modified
- **Availability:** To ensure that the system resources are available to authorized clients when needed
- **Nonrepudiation:** To avoid false denial of involvement in a communication

The following security threats are considered:

- **Denial of Service** – attack on availability
- **Interception** – attack on confidentiality
- **Tampering** – attack on integrity
- **Fabrication, Replaying** – attack on authenticity

A model for the web security has to be build that can implement each of these security services.
This model uses concepts such as public key cryptography, digital signatures and authentication protocols and one of the main standards in web security: SSL (Secure Socket Layer).

3.2 Security Model for the home banking application

In this sub chapter the security model for the home banking application is explained. At first it is important to understand the concept of a session. When a client accesses the home banking application a session for the client is created. This session is used to keep temporary information about the client while he is using the application (information about user id and user authentication). The session has an expire time that closes the session if the client hasn't been requesting the application for a predefined period of time. Another way to close the session is by closing the browser window. Furthermore the application has a log off functionality so the user himself can close the session when he is finished using the application.

Following is a description of each of the security services in the model. The model created involves several modules related to the principles of security services mentioned above, such as access control and authentication, audit trail, confidentiality and integrity etc.

Access Control and Authentication

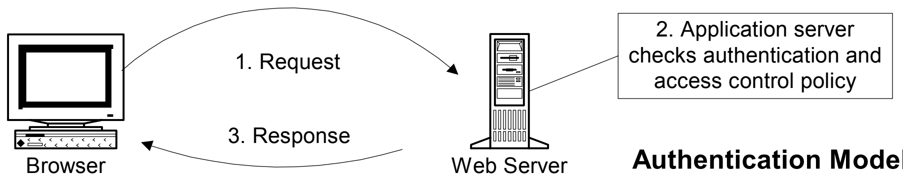
The process of authentication is to confirm your identity.

Both the client of the bank as well as the home banking application has to authenticate themselves to each other. In this thesis the server authentication is done with the use of a certificate and the client authentication is done with a user id and password.

There exist several ways to implement the access control and authentication. Each depending on the demands of the application. In the following several different authentication models are discussed. The purpose of this is to find a model that fits the home banking application well.

Authentication Model 1:

The most simple way is to configure the application server to handle everything. The application server has a good and logic way of dealing with the access control. Basically you tell it which resources, like directories, files or even other services that should be accessible by which roles. Then you add users with their passwords and describe which users have access to which roles. This is a good solution in systems where the pages/files easily can be divided between certain types of clients, normal users, administrators etc.


1. Request
2. Application server checks authentication and access control policy
3. Response
Browser
Web Server

Authentication Model 1

Typically in real home banking applications the authentication is controlled by the application itself or an authentication service outside the application, so it can be customized for specific needs. In these cases the Authentication Model 1 doesn't provide enough flexibility.

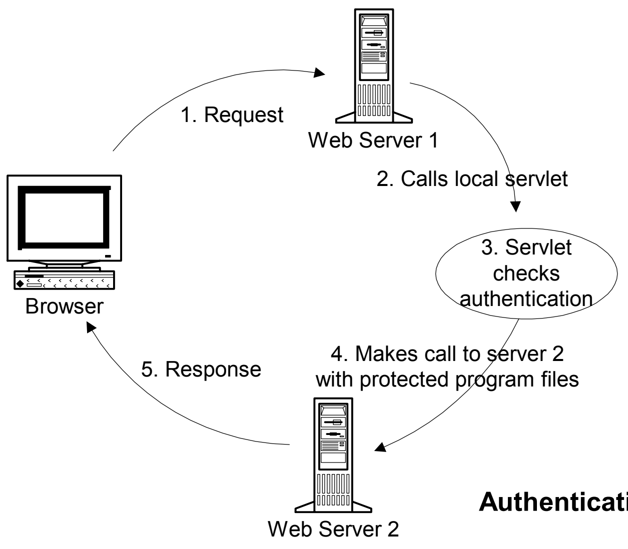
In my application I want to associate groups of functionalities according to their importance defined in the business rules.

Since each function is strongly related to one or more web pages, one solution would be to set an authentication level for each page. The application would then check the authentication level at each request.

Another approach is to have one entry point to the application where each request is compared with the authentication and access control policy. One way of achieving this is to simulate a firewall. This approach is explained in authentication model 2.

Authentication Model 2

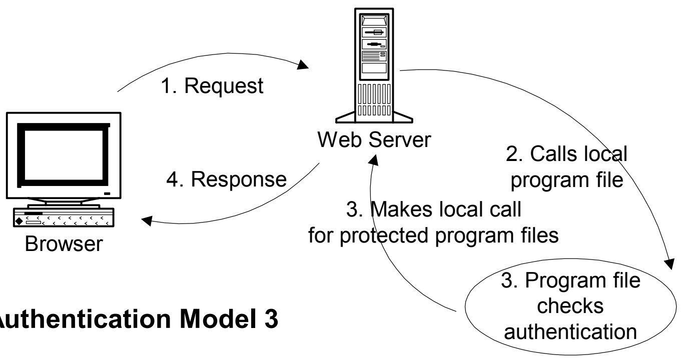
The idea is to have two servers running. Only one server is accessible from the outside and contains the entry point for the application. At this point the authentication check is applied. The second server contains all the files for the application that need to be protected. These files can only by accessed by the first server.


1. Request
Web Server 1
2. Calls local servlet
3. Servlet checks authentication
Browser
4. Makes call to server 2 with protected program files
5. Response
Web Server 2

Authentication Model 2

Authentication Model 3

Yet another solution and *the one used in this case study* is to hide all files from the outside except one that has the entry point for the application and only let this program file be able to access the application files. This is shown in the figure below.


1. Request
Web Server
2. Calls local program file
4. Response
3. Makes local call for protected program files
Browser
3. Program file checks authentication

Authentication Model 3

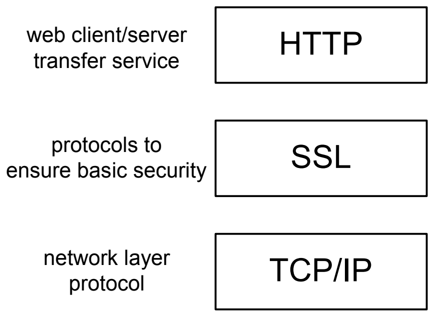
Using this last solution I have the possibility of customizing my own authentication and access control policy and I can implement the one explained above where I associate groups of functionalities with different importance. For each group of functions I have an authentication level. Each level of authentication can have its own way of implementing the authentication. For example I can set some simple functionalities like getting the balance, personalizing the accounts etc. with authentication level 1 and it only requires a user id and a password to access this level. The second level I associate with functions like transfer, payment etc. and I want an increased authentication at this level. In this project the different levels all use static passwords stored in a local database, but the model developed can easily be expanded to support more advanced authentication models at different levels. I could for example implement a model where authentication level 2 is based on smart cards[25] or dynamic password devices.

Audit Trail

An audit trail service is set up for the home banking application. The idea is to log enough information about the client so that an administrator at a later time can get a clear picture of the actions done by the user. This is done by tracking every operation done by the user. Since the logging information depends on the functions and their parameters, this is done inside the application.

HTTPS

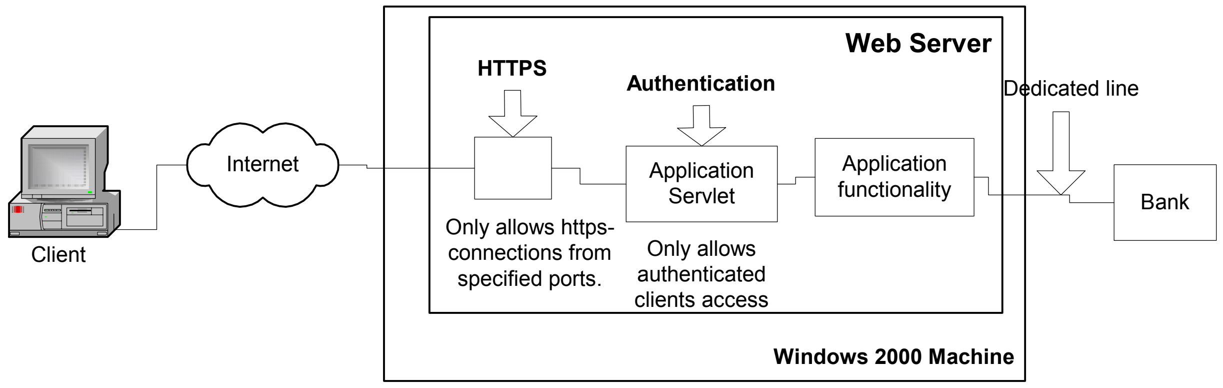
To ensure a secure connection between the client and the home banking application an HTTPS connection is set up. The HTTPS implements a SSL architecture. As shown on the figure below the SSL is placed between the HTTP layer and the TCP/IP layer. The SSL implements several protocols to ensure the secure connection.


web client/server transfer service
HTTP
protocols to ensure basic security
SSL
network layer protocol
TCP/IP

It implements two services for SSL connections:

- **Confidentiality** –a shared secret key is used for conventional encryption
- **Message Integrity** – another shared secret key is used to form a message authentication code (MAC)

The SSL uses **SSL Handshake Protocol** to let the server and client authenticate each other and to negotiate an encryption and MAC algorithm and cryptographic keys which are used to secure the data sent in a SSL record.


Web Server
HTTPS
Authentication
Dedicated line
Internet
Client
Application Servlet
Application functionality
Bank
Only allows https-connections from specified ports.
Only allows authenticated clients access
Windows 2000 Machine

Availability

The availability security is about making sure that the service is always available. Therefore one of the purposes is to avoid a system crash down. The system has to be able to prevent attacks on availability. A typical attack on a web service is a hacker trying to provoke a system break down (denial of service attack) by sending large amounts of requests to the web server. This could cause the web server to go down if it can't handle to many simultaneous requests. To prevent this from happening several measures could be taken to improve the web servers ability to handle large amounts of simultaneous requests. This can be done by setting up several servers to share the load of the requests. Another way to implement a dynamic expire time for the sessions. When the number of requests goes up, the expire time goes down. The application itself will thereby automatically adjust the load of requests.

In the home banking application no particular measures have been taken to provide an availability security service. Though the home banking application is built to ensure that no requests goes to the bank if either the web server is not running (the service is unavailable) or if the local home banking database is down or returns an error. For each request from the client the operation is logged in the local database and if the logging wasn't possible the request never goes to the bank. Instead an error message is returned to the client.

A systems sensitivity to input data depends very much on the implementing platform. In this thesis the Java-platform is used. Since the Java APIs provide access to all commonly used functions you rarely need to let a shell execute commands with user-supplied data. This makes it more secure than for example CGI (Common Gateway Interface)[19]*). Many CGI scripts, if not carefully coded, may use the command shell to execute OS commands. So a creative hacker can make a script to remove all files on the server, mail the server's password file to a secret account, etc.

*) CGI is a method by which a web server can obtain data from (or send data to) databases, documents, and other programs, and present that data to viewers via the web.

4. Home Banking Design Model

In this chapter the model for the development of a home banking application is presented. The model is described by several steps going from the informal requirements specification until the implementation.
Whenever possible the steps are explained so they can be performed in an automated way and thereby providing a basis for implementing a generation mechanism.

4.1 Modelling a Web Site

This model is inspired by the OOHDM model[9] which consists of the following 5 steps:

1. Requirements Gathering
2. Conceptual Design
3. Navigational Design
4. Abstract Interface Design
5. Implementation

This model is based on an object-oriented structure which is also the basis for the implementation of this project.

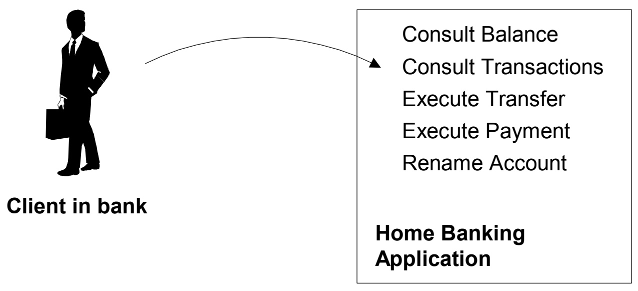
As mentioned in the introduction a web site can be split up into three parts: the functionality, the navigation and the layout. This OOHDM model follows this way of separating the concerns.

UML is the standard for modelling object-oriented systems and is therefore used in the modelling of the navigation and layout presentation. This model is based on ideas from [10][11].

4.2 Requirements Specification

As in all applications, both software and web applications, the first task is to describe the requirements of the application.

The requirements specification is the base used to describe the conceptual model. The requirements depends on the users and the tasks they need to be able to perform on the system.



Users and tasks

For the example system we derive the following users and tasks:

The **users** of a home banking application: **clients of the bank**

Tasks (these were derived from the informal description in chapter 1):
Consult Balance – shows the balance for a given account
Consult Transactions – shows a list of transactions for a given account
Execute Transfer – a money transfer from one account to another
Execute Payment – a service payment
Rename Account – applying a personal name for an account

4.3 Functionality Description - Conceptual Model

In this part of the model the functionality is described and a conceptual model is built. There are several important matters to take into account when defining a function in a home banking site. For each function request the authentication and access control for that particular function has to be checked before it is executed. The only thing needed is to define the authentication level for each function. Furthermore the logging data for each function has to be specified. This is done at the logging specification.

The steps in the functionality description are the following:

- Definition of Functions - define and group the functions from the tasks defined in the requirements specification.
- Access Specification - apply the authentication level for each function
- Logging Specification – define the parameters to be logged for each function
- Database Modelling – the database is modelled and a conceptual model is defined

4.3.1 Definition of Functions

This part of the model defines the step from the informal functionality description to the formal specification of the requirements.

Operations in the bank are seen as transactions. The home banking functionality is a reflection of the banking functionality. Thus each function in the home banking application will be a direct mapping of one or more transactions in the bank, unless it is a function related to the application itself. E.g. the Rename Account doesn't exist in the bank and therefore this functionality might need database storage in the home banking application. Furthermore the functions are divided into consulting functions and operation functions. A consulting function only requests data e.g. consulting the balance, while an operation function tries to execute a transaction which changes the accounts, e.g. a money transfer.

The full description of a function is then defined by its input and output data, which group it belongs to, the authentication level it has and the logging information needed. Each of these steps are now shown for the example system.

The first step is to describe each function by its input and the resulting output and which group it belongs to:

Bank specific operations:

Consulting functions:
consultBalance : ClientId × AccountId→ Balance
consultTransactions : ClientId × AccountIdSource → Transactions

Operation functions:
execTransfer : ClientId $\times$ *AccountIdSource* $\times$ *AccountIdDest* $\times$ *Amount* $\rightarrow$ *Transfer*
execPayment : ClientId $\times$ *AccountIdSource* $\times$ *PaymentId* $\times$ *Amount* $\rightarrow$ *Payment*

Application specific operations:

Personalization functions:
accountNewName : ClientId $\times$ *AccountId* $\times$ *Name* $\rightarrow$ *Accounts*

4.3.2 Access Specification

For this application we want to relate the authentication and access control to each bank operation. If a user tries to execute a command which has a higher level than he is authenticated to he will be presented an authentication page for that level before the command is executed.

At this step in the model we describe the authentication level for each function according to the sensitivity described by the business rules for the application.

For the example system the following levels are set:

Consulting functions:
consultBalance **: level 1**
consultTransactions **: level 1**

Operation functions:
execTransfer **: level 2**
execPayment **: level 3**

Personalization functions:
accountNewName **: level 1**

4.3.3 Logging Specification

Ideally the logging functionality of a home banking site is a plug on, like an extra service.
In real home banking applications there exists several logging functionalities. The two main logging parts are the logging for the audit trail security service and the other is logging for development purposes. Furthermore the bank itself logs everything in the communication between the bank and the home banking application. If some doubt from a client or a bank administrator appears the logging from the bank is compared against the logging from the home banking application.

In a real home banking application every operation done by the client is logged in the home banking application before any request to the bank is made. If there was a problem with the logging the home banking application doesn’t contact the bank, but returns an error to the client. This ensures that the system is running correct and that everything will be logged.

In this project the logging for the audit trail security service is implemented. The logging information is directly related to each function in the home banking application.

For each operation the client id, function authentication level and a time stamp is logged.

In this part each function is described with the logging parameters.

Consulting functions:

consultBalance : AccountId $\rightarrow$ *BalanceOk*
consultTransactions : AccountId $\rightarrow$ *TransactionListOk*

Operation functions:

transfer : AccountIdSource × *AccountIdDest* × *Amount* $\rightarrow$ *TransferOk*
payment : AccountId × *PaymentId* × *Amount* $\rightarrow$ *PaymentOk*

Personalization functions:

accountNewName : AccountId × *Name* $\rightarrow$ *AccountNewNameOk*

4.3.4 Database Modelling

The main purpose of the database related to the home banking application is to keep user information, like user name, password, user status, etc.. Concepts that are related to the home banking application. Furthermore the database is used to keep information about the name of accounts, logging information and error codes with messages, etc..

The modelling of the database is done following an Entity Relation model defined by the following 6 steps:

1. Choose Entities
2. Find attributes for entities
3. Normalization
4. Create ERD
5. Remove many-to-many relationships
6. Redraw final ERD

Following these steps a conceptual model describing the functionality seen from the client as well as from the internal functionality needed for the home banking application to run is build.

For the example system the following is modelled:

Entities

Home banking application

Client – contains the login data about the client

Account – contains additional information about the accounts that doesn't exist in the bank

OpLog – contains the logging of the operations related to accounts

OpType – contains the different home bank operation types

Error – contains the different error types

Bank application

Client – contains the data about the client in regards to the bank

Account – contains the information about the account

Transaction – contains information about transactions related to an account

Attributes and keys for entities

The key attributes are underlined

Home banking application

Account	
<u>account id</u>	integer
name	string

Client	
<u>client id</u>	integer
<u>level</u>	integer
password	string
last login	timestamp

OpLog	
<u>op id</u>	integer
account id	integer
date	timestamp
detail	string
status	integer

OpType	
<u>op id</u>	integer
type	string

Error	
<u>error id</u>	integer
message	string

Bank application

not all the entities are shown for the bank application – only the once with relevance to the example system

Account	
account id	integer
balance	double

Transaction	
transaction id	integer
amount	double
account Src	integer
account Dst	integer
description	string

Client	
client id	integer
account id	integer
name	string
address	string

Normalization

Now it is verified if the database is normalized, that is if it lives up to the 5. NF.

1NF

Definition: Non-key attribute in a table should be functionally dependent on the whole key.
This is true for all entities.

2NF

Definition: 1NF, plus it includes no partial dependencies; that is, no attribute is dependent on only a portion of the primary key.
This is true for all entities.

3NF

Definition: 2NF, it contains no transitive dependencies.
This is true for all entities.

4NF

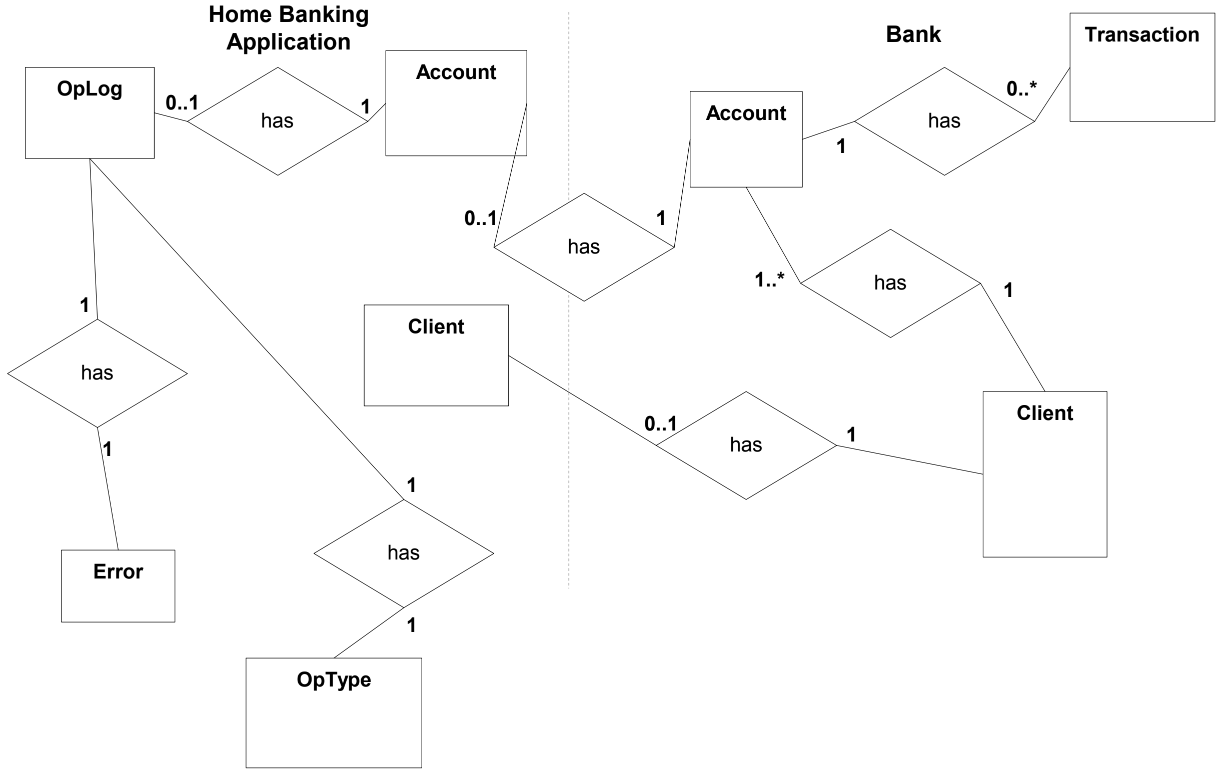
Definition: There can not be 2 or more independent multi-value facts about an entity.
This is true for all entities.

5NF

Definition: No table can be build from a join of smaller tables.
This is true for all entities.

ERD

The following entity-relationship diagram is derived:

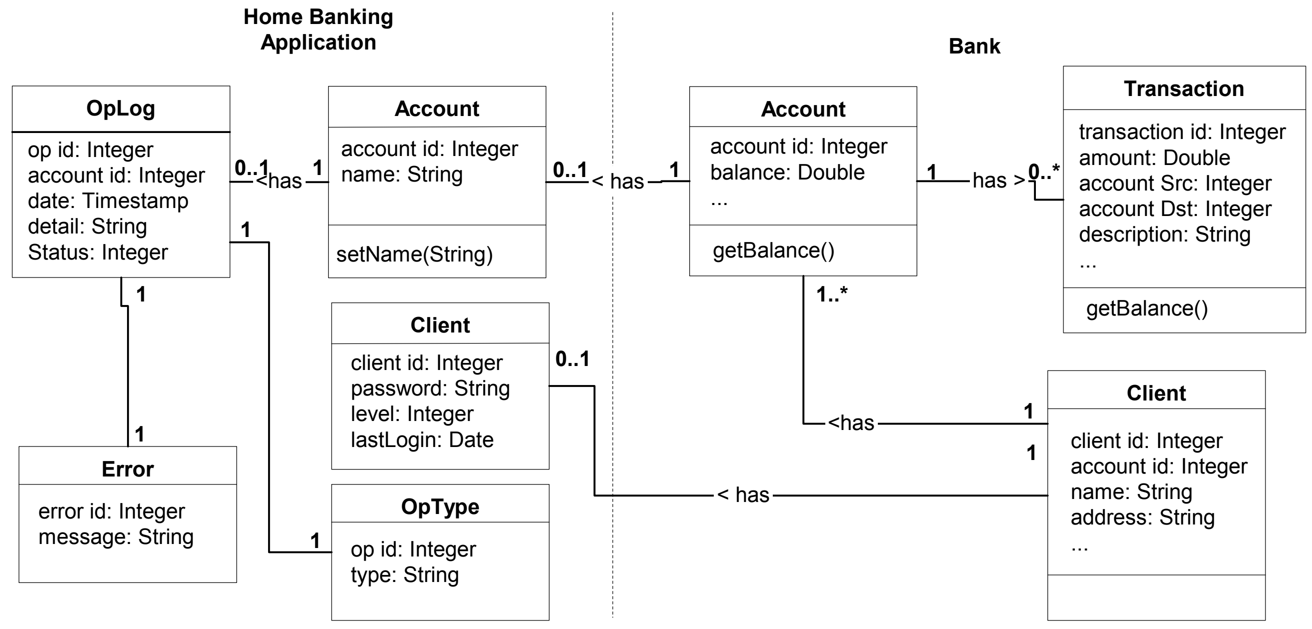


Remove many-to-many relationships

There exist no many-to-many relationships.

The conceptual model is created from the ERD by adding the functions where they affect the table. The relationship symbols are not shown. The following conceptual model is derived for the example system:

Conceptual Model for example system:



4.4 Navigational Design

In this part the navigational modelling is described.

This part of the model focuses on the navigational design of the web site. The navigational design describes which parts of the conceptual model is to be seen and how the user can navigate to see them. Thus the purpose of this part of the model is to show how the navigational model can be systematically developed from the conceptual model.

The navigational model in a web application has to allow the user not only to browse through the information being shown in the web application but also to operate on it.

The navigational model is built from the conceptual model and is seen as a view over the conceptual model.

It is defined by two steps:

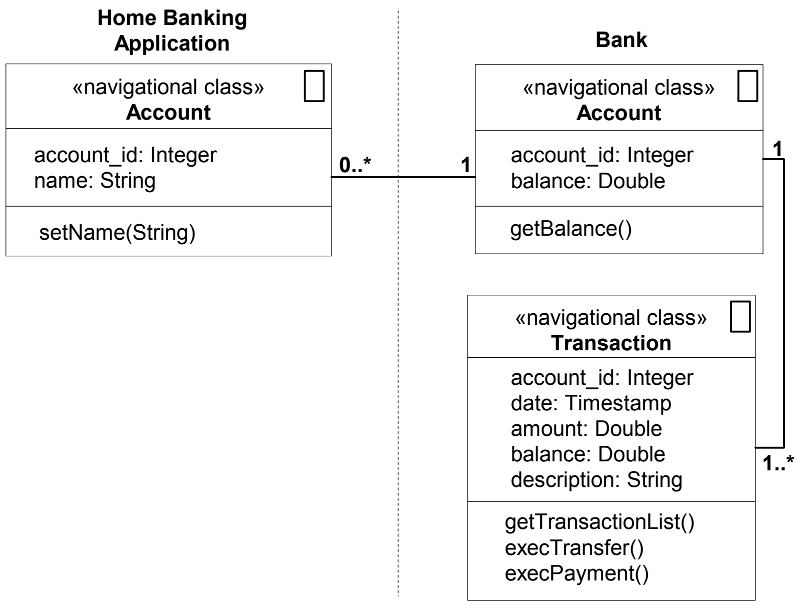
The navigational space model shows the classes of the conceptual model that can be visited
The navigational structure model defines the navigation of the application.

4.4.1 Navigation Space Model

The navigation space model shows the objects that can be visited by direct navigation.

The navigation space model is derived from the conceptual model with the following steps:
- All classes that are relevant for the navigation are included in the navigation space model.

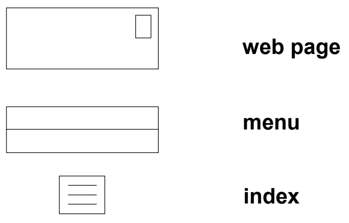
- All attributes in the conceptual model map directly to attributes in the navigation space model if they are relevant



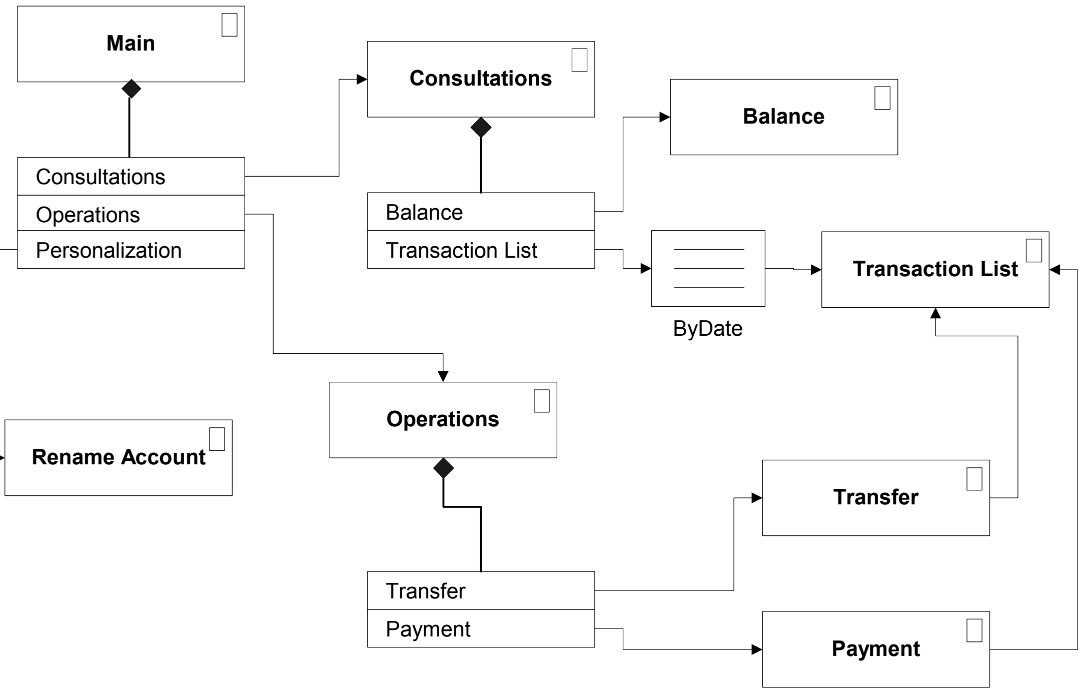
4.4.2 Navigational Structure Model

Next is created a navigation structure model that shows the navigation between the navigational classes. This model is based on a simplified version of [10] and [11].

UML extension symbols used in the Navigational Structure Model:



Navigational Structure Model for example system:



4.5 Layout Design

The layout of a page is often very closely connected to the graphical design of the page. The layout mainly deals with the placement of the content for each page, while the graphical design determines the font type, size and colour, background colours, pictures, logos, etc. It is the more artistic part of a web site and the people involved with the graphical layout tend to give more value to the visual experience of the site thinking less of the functionality behind. Designing a web site is a balance between being original in the use of graphics and layout of the page while giving a good user experience and making sure the site shows the right profile and identity of the company. Furthermore the layout and graphical design is dependent on the restrictions set by the browser and technology used. For example creating the whole site as a Flash animation would give more freedom to graphical designer, but might give some other problems with functionality.
Generally the more functionality and size a site has, the less freedom the graphical designer has.

In a home banking site the functionality often has the highest priority and the graphical layout is very restricted. Most of the pages contain either tables with text and values and/or pages with fill-out forms.

Basically the content of each page is defined from the functionality of the page and then has to be mapped to the page using some layout criteria's.

To define a method for laying out the pages we first need to define the different types of pages and their elements. There exists and is constantly being developed a lot of technologies and techniques to aid in the structuring and graphical presentation of the page as well as providing methods for creating dynamic pages. As the foundation, HTML defines the language used to layout the content on the page. HTML in itself is purely static, but using technologies such as JavaScript pages can change the content dynamically without requesting the web server for a new page. Furthermore there exists several plugins (like

Flash[22]) to the browsers providing additional possibility of creating more animated and graphically attractive web sites.

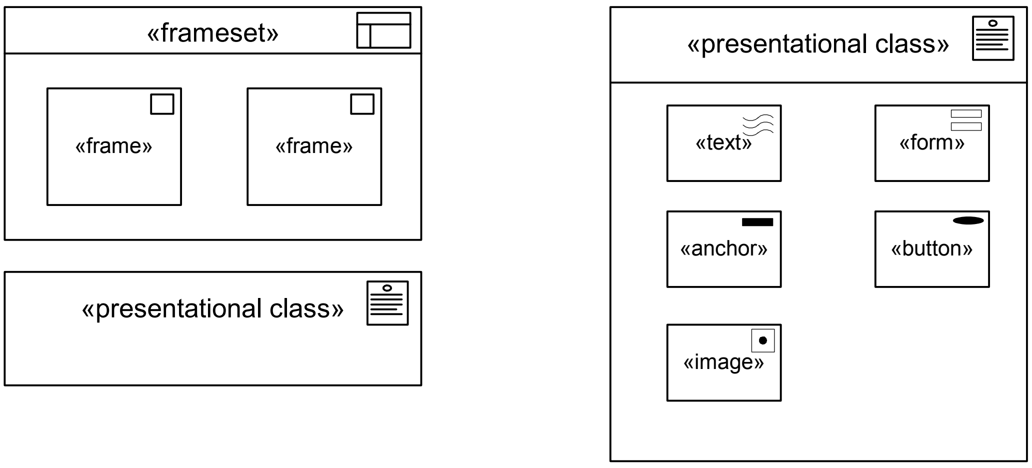
To formalize the layout a presentational model is created. It shows the layout of the

At the top level of the presentational model a frameset is described. A frameset splits up the page in different areas to arrange presentational objects, but may contain other nested framesets.

4.5.1 Presentational Model

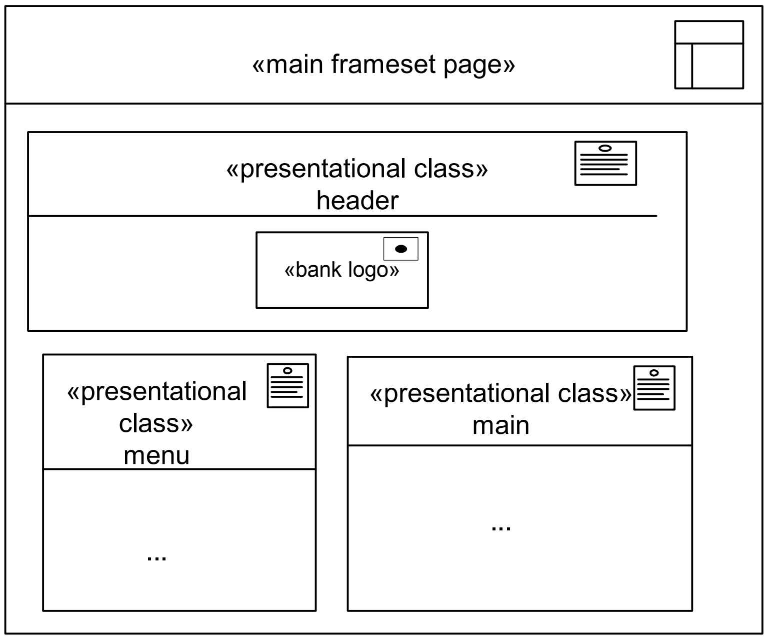
The presentation model is based on framesets. This is where the pages are split up in frames that can be directly implemented with HTML frames.

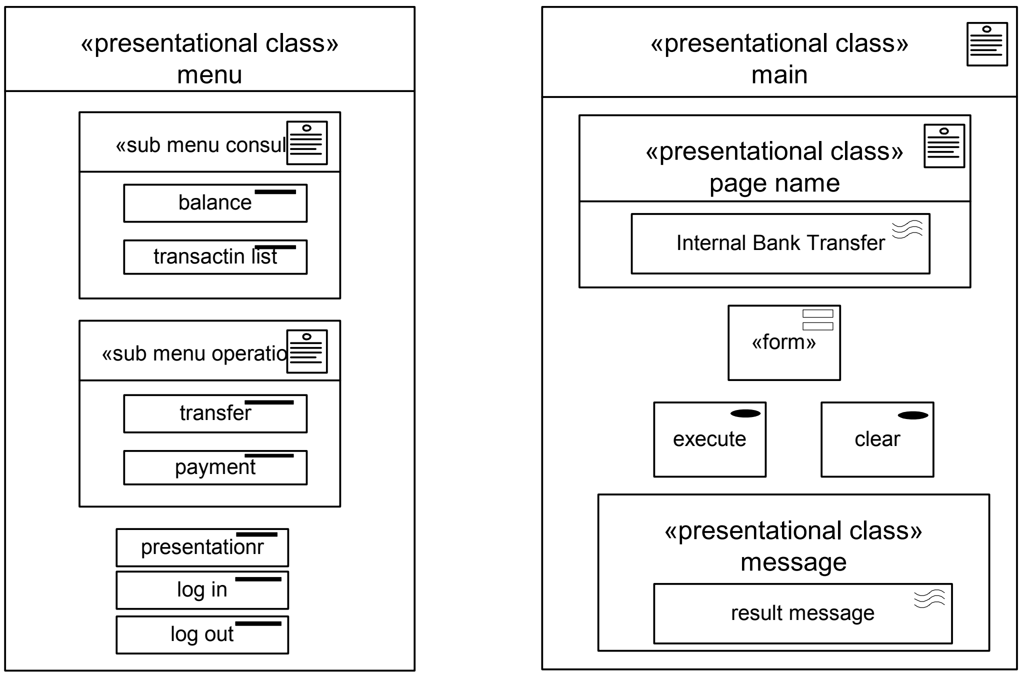
The following symbols are introduced to construct the presentation model:



The presentational class contains the elements or groups of elements used on the HTML pages.

For the example system the following presentation model is derived – only the main presentational class for the transfer page is shown. The presentational classes for the other pages are very similar because they follow the same structure. E.g. the payment presentational class has the same layout only the title and form is different.


«main frameset page»
«presentational class» header
«bank logo»
«presentational class» menu
...
«presentational class» main
...



4.5.3 Applying CSS – Style, Colours, Fonts, Images

In the last step of the Home banking design model the colors, font parameters and general layout parameters are applied.

In a home banking site functionality is one of the main concerns since we are dealing with banking operations. And the pages in a home banking site tend to have few pictures and a lot of data to be presented. One technology that definitely aids in separating the graphical presentation from the content in these cases is CSS – Cascading Style Sheets.

CSS is a tool built into the latest browsers that gives the possibility of reusing and gathering information about styles, fonts, colors, margins, etc., for the text, tables, form elements, etc..

Different pages can load the same CSS file that provides a way to describe colour, style, etc., for fonts and table elements, etc., which are the elements mainly used when a lot of data and/or a lot of pages has to be presented in an organized fashion. In this way changing the appearance of the page is centralized and very easy. Furthermore CSS-files can load each other in an object-oriented way so one CSS file can inherit styles from another. By the use of CSS a great part of the graphical layout can be dealt with at the top level separating it in a simple and powerful way from functionality.

The CSS files are organized according to the views that use them.

CSS applies the following:

font parameters (size, font color, face etc.)
colors (background, layers etc.)
table layout (margins, padding)

Typically we want the same type of layout for similar pages or as a minimum make sure each page have some sort of identical representation. This is a way of improving the user experience, ease the navigation and facilitate the recognition of where the user is.

Applying CSS

CSS styles are applied to each element in the presentation class as well as each type of presentation class.

For the example system we get:

First the three main pages that is separated by the frameset:

«presentational class» **header**:
style sheet cssHeader

«presentational class» **menu**:
style sheet: cssMenu

«presentational class» **main**: (same for all main pages like, balance, transfer, etc.)
style sheet: cssMain

Then the elements on the main page:

«presentational class» **page name**:
style sheet: cssPageName

«presentational class» **page name text**:
style sheet: cssPageNameText

«presentational class» **message**:
style sheet: cssMessage

«presentational class» **message text**:
style sheet: cssMessageText

«presentational class element» **button**: (same for both **execute** and **clear button**)
style sheet: cssButton

Then follows the elements on the menu page:

«presentational class» **sub menu**: (same styles for **consult** and **operation menu**)
style sheet: cssSubMenu

«presentational class element» **anchor**: (same for each link in the menu)
style sheet: cssAnchor

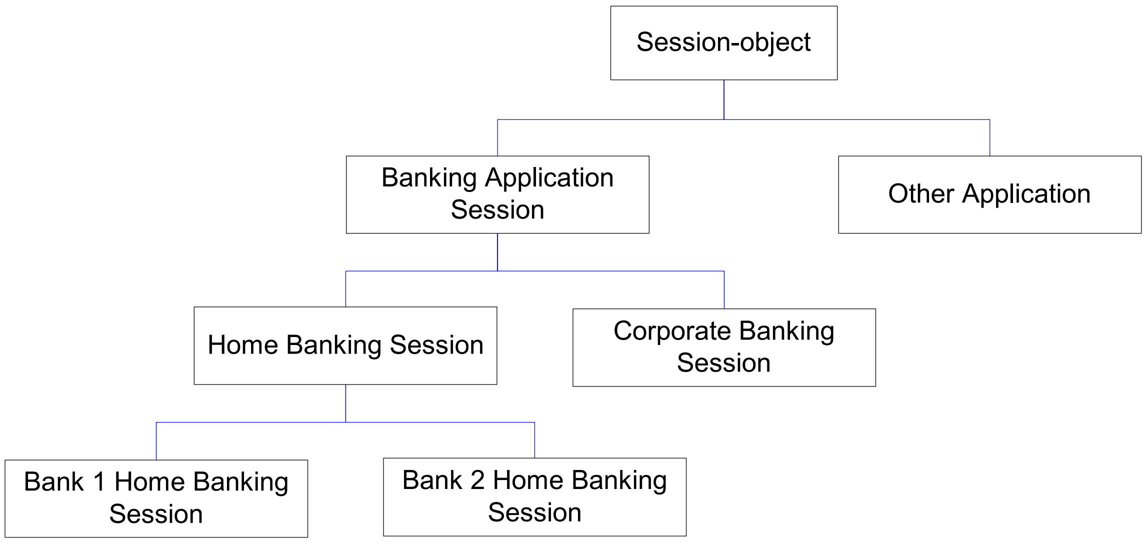
5. Implementation

This chapter focuses on the implementation of the home banking site. The first sub chapter (5.1) describes the chosen technology Java, JSP and servlets. The second sub chapter (5.2) shows how the JSP and servlets can be combined to create an implementation model for the application. In the third chapter (5.3) an implementation of this structure is then introduced along with the file structure of the application. In sub chapters 5.4, 5.5 and 5.6 the step from the model to the implementation is described. This is followed by chapter 5.7 that explains the implementation of the authentication model defined in chapter 3.2. Chapter 5.8 describes the implementation of the Bank Simulator and finally chapter 5.9 contemplates on some of the difficulties and decisions taken during the implementation process.

5.1 Choice of technology – Java technology (JSP and Servlets)

For this case study Java technology has been chosen as the implementation platform. With its object-oriented structure it provides an efficient building ground for reusability in a banking application environment.
Imagining a company that creates several applications for several banks. By using the Java structure and organizing the code well several advantages are clear.
As an example of reusing and organizing code let's take a look at the different levels. On the top level exists objects used by all the applications like for example a session-object with general functionality. This objects share general functionality used by several different applications. This object is extended and functionality added for the next level where different applications add the extra functionality needed. These applications can again be split up and extended. In the lowest layer several different banks use the same applications and thus extend the application objects (see figure).

One possible organisation of the code for an object used throughout the applications e.g. a session-object could be like this:


Session-object
Banking Application Session
Other Application
Home Banking Session
Corporate Banking Session
Bank 1 Home Banking Session
Bank 2 Home Banking Session

Furthermore using Java you have a robust underlying structure. Robust because it avoids using pointers and other means of addressing memory directly. It also checks array boundaries, etc. and thereby eliminates the risk of overwriting memory and corrupting data that could cause a system shutdown. That along with the exception handling aids greatly in creating secure programs when dealing directly with input data, which is important in a home banking application.

Java Servlet

A servlet[6] is a Java web component that can create dynamic content. By dynamic content is meant that the output to the web browser can change dynamically according to the calculations done by the servlet and the input data received. It runs in a *servlet container* which is part of a web server that handles network services like a request, response, etc.. It can replace the need for gateway programs like CGI-programs and other similar technologies that provide the interface between the web server and external programs. Servlets are invoked through URL invocation.

A typical sequence of events for a servlet request could be like this:

1. A client/(web browser) makes a HTTP-request to a web server
2. The request is received by the web server and handed over to the servlet container
3. The container uses its configuration to determine which servlet to invoke and calls it with objects representing the request and response
4. After that it uses the request parameters to determine the remote user and gather the data that was possibly sent with the request. It then performs the functionality it was programmed for. E.g. call a Java Bean for additional functionality. And finally generates the data needed to send back to the client. The data is sent back with the response object.
5. When the servlet is finished processing the request, the container flushes the response and returns control back to the web server.

JavaBean

A JavaBean[5] is a self-contained Java software component. It provides a structure for building reusable Java components. The structure follows certain *design patterns* when naming JavaBean features and can therefore be used in conjunction with JavaBean builder-tools for easier implementation. It works efficiently together with Java servlets to keep parts of functionality outside the servlets and provide data for a JSP-call (see chapter 5.2). In the implementation structure used in this thesis JavaBeans are used to contain the functionality of the home banking application.

JSP – Java Server Page

A JSP[7] is a page, much like an HTML page, that can be viewed in a web browser. However, in addition to HTML tags, it can include a set of JSP tags and directives intermixed with Java code that extend the ability to incorporate dynamic content in a page. One of the main benefits of JSPs is that, like HTML pages, they do not need to be compiled. The web designer simply writes a page that uses HTML and JSP tags and puts it on their web server. The web designer does not need to know how to define Java classes or use Java compilers.

JSP pages can access full Java functionality in the following ways:

by embedding Java code directly in scriplets in the page
by accessing Java beans
by using server-side tags that include Java servlets

A JSP page is an extension of a Java servlet and provides a good way of holding the layout of a web page. It can help avoiding mixing the functionality with the layout.

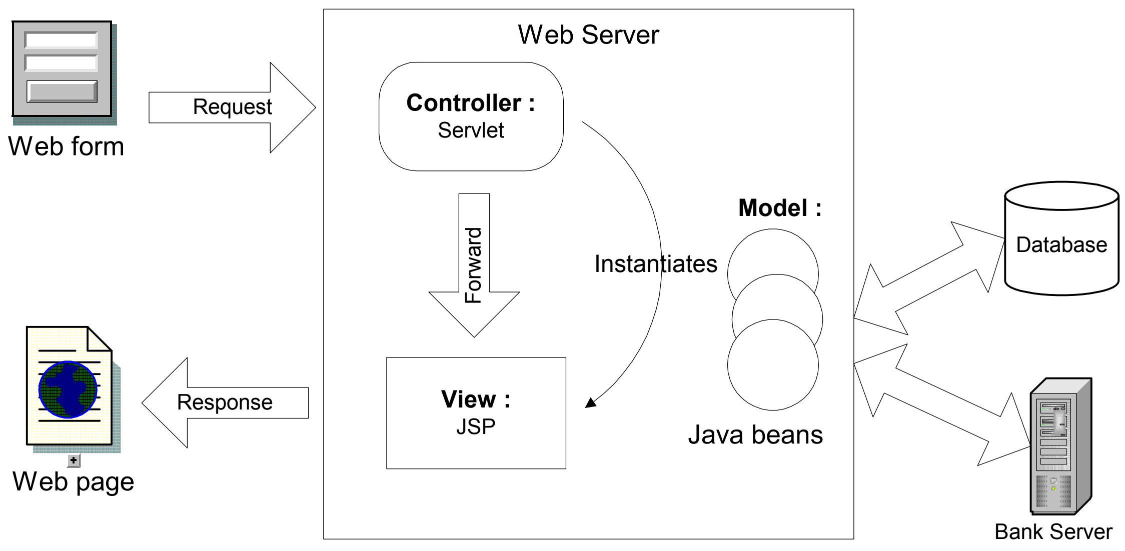
5.2 MVC - Model View Controller (Java Server Pages Model 2)

Using Java Server Pages together with Java Servlets combined in the Model View Controller model provides a good framework platform for the model. It effectively helps in separating layout, navigation and functionality. It implements a good structure for a bank site.

Model – The model is the main functionality of the web application. It consists of Java Beans that contain the business logic and connectivity with other resources like databases, web services etc. (It also keeps the state of the web application)

View –The view together with the CSS file(s) describes the layout of the page. A view is basically a JSP pages that contains the html used to build the pages. It furthermore contains some Scriplet code (could be replaced by tags) that is mainly used to catch the data to be shown from the beans.

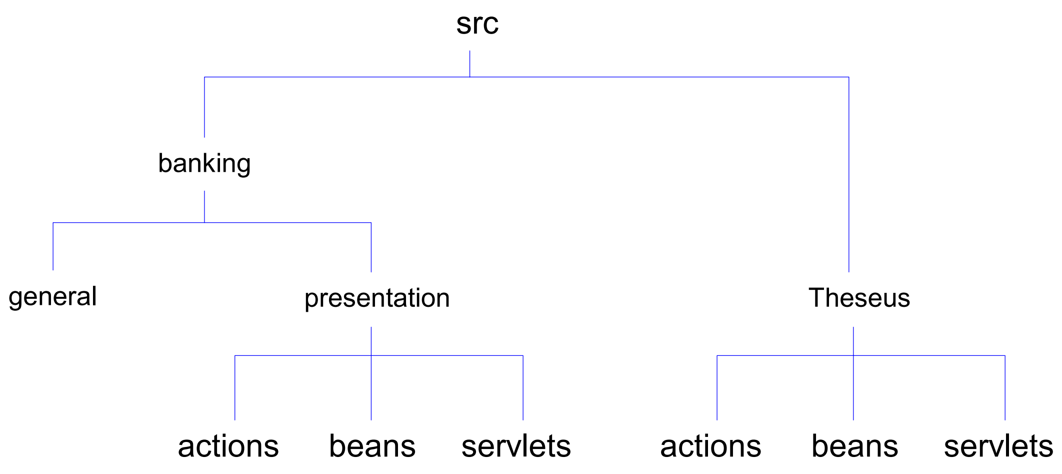
Controller – The controller takes care of receiving the form data, instantiating the right model objects (beans) and redirecting to the correct view page.



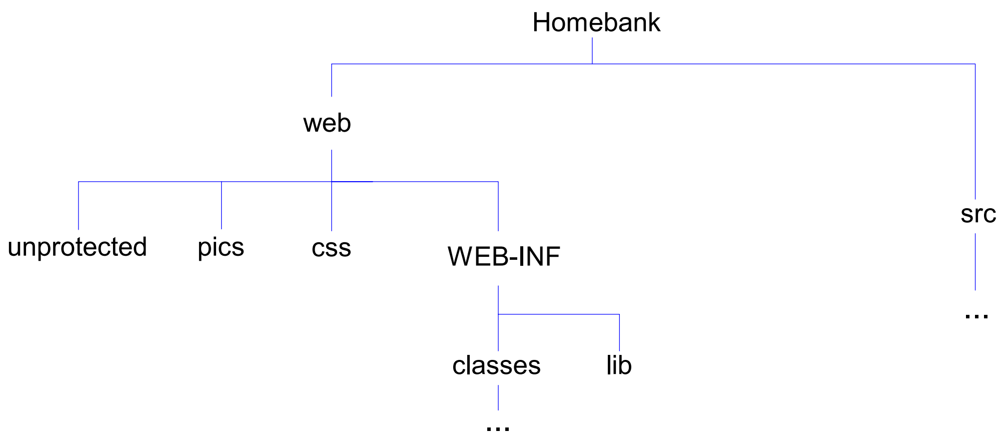
5.3 Implementation Structure

This chapter shows the implementation structure of the home banking application. As a base for the implementation is used an already implemented simple example of the MVC model called Theseus[17]. This implementation has a controller servlet which has been changed for this application, some extra classes for instantiating the bean and an example.

The Java classes has the following structure:



The whole directory structure for the application:



The main following types of files exist:

Layout files:
css/stylesheet.css, balance.jsp, transfer.jsp, etc.

Functionality files:
GenericBean, BalanceBean, TransferBean, PaymentBean, etc.

Navigation files:
ConsultationAction, OperationAction, etc.

Layout file explained:

In the following the main points of balance.jsp file is shown and explained. The HTML tags are not explained.

```
1  <%@ include file="/loadBean.jsp" %>
2  <% BalanceBean b = (BalanceBean)in.readObject(); %>
3
4  <html>
5  <head><title>balance</title>
```

```
6  <%= Global.getStylesheet() %>
7  </head>
8
9  <body class="cssMain">
10
11 <div class="cssHeader">&nbsp;:: Consult the balance of your accounts</div>
12
13 <div class="cssPageName">
14 <table cellpadding="0" cellspacing="0">
15 <form method="post" action="/ConsultBalance.do?command=GetBalance&auto-
16 populate=true"><tr>
17 <td class="cssPageNameText">Active account:</td><td class="formField">
18 <%= b.getAccountsSelect("selectField") %></td>
19 </tr></form></table>
20 </div>
21
22 <br><br>
23 <table class="balanceBorder" cellpadding="1" cellspacing="0" border="0"><tr>
24 <td class="formNameBalance">Balance:</td><td class="formValueBalance">
25 <%= b.getBalance() %>&nbsp;€</td>
26 </tr></table>
27
28 <br>
29
30 <div class="message">
31 <%= b.getMessage("okMessage","errorMessage") %></font></div>
32
33 </body>
34 </html>
```

The line numbers are added only to help describing the functionality. They don't exist in the real balance.jsp file.

line 1: import a file with a general structure for loading the bean
line 2: the bean containing the functionality for the page
line 6: loads the style sheet file for this document. The information is stored
 in Global to make it easy to change for all or individual files
lines 11,13 and 30: generally the <div> tag is used to surround the view parts.
 With a <div> several parameters like font attributes, padding, etc. can
 be set for the area the tag surrounds
lines 18,25 and 31: shows how methods from the functionality bean is invoked
 to get the values calculated in the functionality part. In line 31 it is shown
 how the styles can still be used even though the element is coming from
 the and that different styles can be used for different purposes. In this
 case we want different colour for different types of messages.

Functionality file explained:

In the following the functions of **BalanceBean** is explained. The BalanceBean extends GenericBean.

BalanceBean() - sets the type of the operation - used for logging
refresh() - called to reset message values etc.
setAccount(String value) - called during population of the class - to set the parameters send
 when a form is submitted

the following methods return the values used on the balance.jsp page:

> **getBalance()** - returns the balance
> **getAccountsSelect()** - returns a select with the accounts belonging to the client
> **getAccountsSelect(String style)** - same as **getAccountsSelect()** but adds a style sheet the select
> **getMessage()** - returns the message, can be either an error message or a success message
> **getMessage(String styleOk, String styleError)** - same as getMessage() but gives the possibility of showing the style

the following functions contacts the bank by invoking a function through the web service

> **getAccountsFromBank(ActionContext context)** - calls bank to get accounts for client
> **getBalanceFromBank()** - calls bank to get balance for an account

Navigation file explained:

In the following the methods of **ConsultationAction** is explained. ConsultationAction extends Action. Each of these methods are called from ControllerServlet and reflect a command on the view (JSP) pages. E.g. GetBalance is a command on the balance.jsp page.

executeCommandShowBalancePage(ActionContext context) – relates to a command on the menu.jsp used to show the balance page without having an account selected
executeCommandGetBalance(ActionContext context) – relates to a command from balance.jsp to show the balance for a given account
executeCommandShowTransactionListPage(ActionContext context) – relates to a command on menu.jsp used to show the transaction list page without having an account selected
executeCommandGetTransactionList(ActionContext context) – relates to a command on the transactionList.jsp to show the transaction list for a given account

Now it is described how this relates to the MVC structure:

Controller:
> **Theseus.servlets.ControllerServlet**: this is the controller servlet in the MVC model
> **banking.presentation.servlets.BankingHook** : is called from the ControllerServlet to make the authentication check

Model:
> **banking.actions.ConsultationAction**: for each function-group there exists an action– this is the action for the consultations
> **banking.beans.BalanceBean**: for each function exists a bean – this is the bean for the balance

View:
> **balance.jsp**: all the view files exist in the Homebank/web directory – this is the view for the balance

MVC-functionality

When the web server is started and the first page request is made an init procedure is called in the Controller Servlet that sets up the Actions and Authentication levels used at each request.
The information is stored in the files: auth-config.xml and action-config.xml.

The purpose of these two files is now explained along with two examples.

auth-config.xml

This file contains the authentication level for each function and the resource (in this case a login file) for each level. This information is used in the Controller Servlet to check that the client is authenticated to the right level when executing a function.

Here is part of the auth-config.xml file for the example system. The whole file is shown in the auth-level implementation step (chapter 5.4)

```
<auth-config>
  <!-- authentication-level for each command -->
    <auth command="ShowBalancePage" level="1"/>
    <auth command="GetBalance" level="1"/>
  ...

  <!—login page for each authentication level -->
    <auth-resource level="1" page="/unprotected/loginLevel1.jsp"/>
    <auth-resource level="2" page="/unprotected/loginLevel2.jsp"/>
  ...
</auth-config>
```

action-config.xml

This file contains the action configuration used by the controller servlet to set up the Action class
for each function. Every group of functions uses the same Action class. It furthermore contains the Bean class for each function as well as the resource that the controller servlet will direct to when a command has been executed.

Here is part of the action-config.xml file for the example system. The whole file is shown in the auth-level implementation step (chapter 5.4).

```
<mvc-app>
    <action name="ConsultBalance" class="banking.presentation.actions.ConsultationAction">
    <bean name="BalanceBean" class="banking.presentation.beans.BalanceBean"/>
    <forward name="ShowBalanceOK" resource="/balance.jsp"/>
    <forward name="GetBalanceOK" resource="/balance.jsp"/>
  </action>
  <action name="ConsultTransList" class="banking.presentation.actions.ConsultationAction">
  ...
</mvc-app>
```

To understand the overall flow of the application the Controller Servlet is now explained. The most important functionality happens in the init function and the service function.

The **init** function is called the first time a request for the Controller Servlet is made.
The **service** function is called for each request for the Controller Servlet.

init function

These are the main things done in the init of Controller Servlet:

- set up local homebank database connection
- auth-config.xml is read and the authentication levels inserted into a Map
- action-config.xml is read and the action classes and references are put into a Map

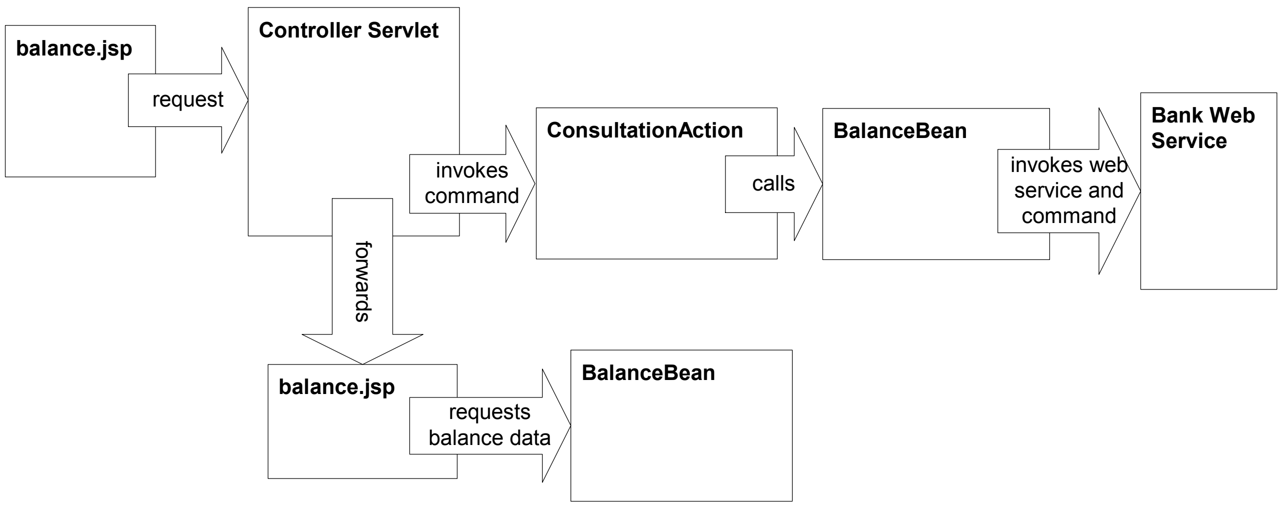
service function

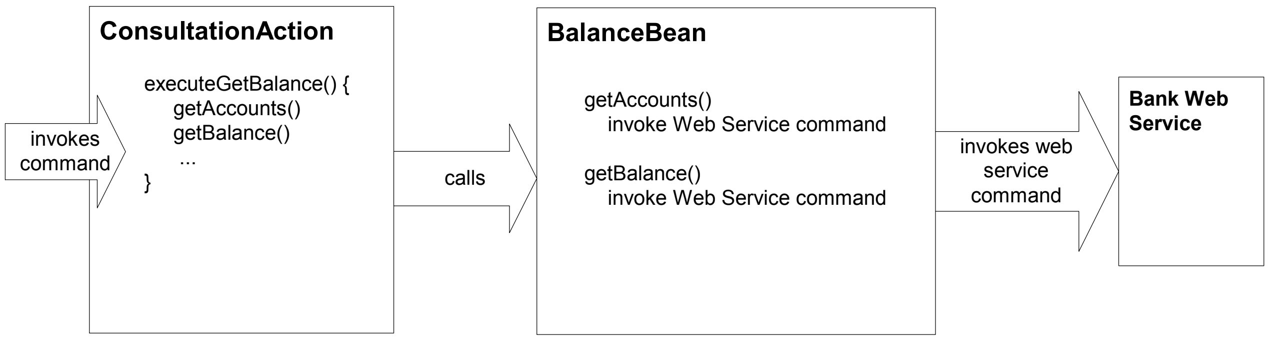
Explanation of main steps when executing the consultBalance command:

1. Browser requests from the balance.jsp page with parameters: command=GetBalance
2. Web Server receives request and redirects to the service function in Controller Servlet
3. The command is verified that it exists
4. The homebank database is checked if it is running so logging for each command can be done
5. If these two criteria are full filled the Controller Servlet checks the authentication level for command by calling BankingHook
6. If client is authenticated for at least level 1 (required for consultBalance) control is returned to the Controller Servlet else it redirects to the login page for level 1
7. Assuming the user is logged in the Controller Servlet reads the command: GetBalance from the query string
8. It furthermore reads the first part of the query string which tells which Action to use (the Action contains the bean with functionality for the command)
9. The Action is instantiated if it wasn't found in either the request, the session or in the application as an attribute. If it wasn't found it is then stored in either the request, session or application for later use
10. Then the Controller Servlet calls a function that "populates" (calls a setter-function in) the BalanceBean with the parameters sent from the page. In the case of consultBalance it is only the account number
11. The parameter values needed for the consultBalance command are now ready and the Controller Servlet invokes an executeGetBalance command in ConsultationAction which is the Action bean belonging to the command consultBalance
12. The ConsultationAction bean calls two functions in the BalanceBean – first the function getAccounts which gets the accounts and then the getBalance which gets the balance
13. The getAccounts and getBalance get their result from the bank. This is done by invoking a getAccounts and getBalance command on the web service provided by the bank. The web service (the bank) contacts the database and returns the accounts and the balance to the BalanceBean
14. The BalanceBean now has the values to be shown on the browser so they return an ok to the ConsultationAction which returns the name of the page that the Controller Servlet will forward to, which in this example case is balance.jsp
15. The Controller Servlet then forwards to the page along with the BalanceBean which contains the data to be shown on the page
16. The balance.jsp page is called and this page calls methods in BalanceBean which returns the accounts and the balance for the requested account

This is now shown in the following diagram with boxes representing Java classes and arrows representing function calls or function invocations.

Example of execution of command: *consultBalance*





The arrows in the figures above only show the call of functions and doesn't tell whether values were returned for each call or not.

5.4 Mapping navigation and functionality

The function groups represent collections of functions with similar functionality. It is therefore practical to keep them together in the implementation.

Each function-group relates to an Action-class:

> **Consultation-group** -> **ConsultationAction**
> **Operation-group** -> **OperationAction**
> **Personalization-group** -> **PersonalizationAction**

The beans represent the functionality of the home banking application.

Each function relates to a Bean-class:

consultBalance -> **BalanceBean**
consultTransactions -> **TransactionListBean**
execTransfer -> **TransferBean**
execPayment -> **PaymentBean**
accountNewName -> **NewNameBean**

Each function inside the balance-group relates to a command:

consultBalance **-> executeConsultBalance**
consultTransactions -> **executeConsultTransactions**
transfer -> **executeTransfer**
payment -> **executePayment**
accountNewName -> **executeAccountNewName**

action-level implementation step

At this step an action xml-file is made that defines an action class for each command. These commands were defined in the functionality description (chapter 4). Each action contain the bean with the functionality for the action and the resources that can be called from the action class when a command has been called.

The action-config.xml for the example system:

```
<mvc-app>
   <action name="ConsultBalance" class="banking.presentation.actions.ConsultationAction">
   <bean name="BalanceBean" class="banking.presentation.beans.BalanceBean"/>
   <forward name="ShowBalanceOK" resource="/balance.jsp"/>
   <forward name="GetBalanceOK" resource="/balance.jsp"/>
  </action>
  <action name="ConsultTransList" class="banking.presentation.actions.ConsultationAction">
   <bean name="TransListBean" class="banking.presentation.beans.TransactionListBean"/>
   <forward name="ShowTransactionListOK" resource="/transactionList.jsp"/>
   <forward name="GetTransactionListOK" resource="/transactionList.jsp"/>
  </action>
  <action name="OperationTransfer" class="banking.presentation.actions.OperationAction">
   <bean name="TransferBean" class="banking.presentation.beans.TransferBean"/>
   <forward name="ExecTransferOK" resource="/transfer.jsp"/>
   <forward name="ShowTransferOK" resource="/transfer.jsp"/>
  </action>
  <action name="OperationPayment" class="banking.presentation.actions.OperationAction">
   <bean name="PaymentBean" class="banking.presentation.beans.PaymentBean"/>
   <forward name="ExecPaymentOK" resource="/payment.jsp"/>
   <forward name="ShowPaymentOK" resource="/payment.jsp"/>
  </action>
  <action name="Personalization" class="banking.presentation.actions.PersonalizationAction">
   <bean name="NewNameBean" class="banking.presentation.beans.NewNameBean"/>
   <forward name="ShowNewNameOK" resource="/newName.jsp"/>
   <forward name="ExecNewNameOK" resource="/newName.jsp"/>
  </action>
</mvc-app>
```

auth-level implementation step

At this step an authentication xml-file is made that defines a level for each command. These levels were chosen in the functionality description (chapter 4).

The auth-config.xml for the example system:

```
<auth-config>
  <!-- authentication-level for each command -->
    <auth command="ShowBalancePage" level="1"/>
    <auth command="GetBalance" level="1"/>
    <auth command="ShowTransactionListPage" level="1"/>
    <auth command="GetTransactionList" level="1"/>
    <auth command="ShowTransferPage" level="2"/>
    <auth command="ExecTransfer" level="2"/>
    <auth command="ShowPaymentPage" level="3"/>
    <auth command="ExecPayment" level="3"/>
    <auth command="ShowNewNamePage" level="1"/>
    <auth command="NewName" level="1"/>

  <!—login page for each authentication level -->
    <auth-resource level="1" page="/unprotected/loginLevel1.jsp"/>
    <auth-resource level="2" page="/unprotected/loginLevel2.jsp"/>
    <auth-resource level="3" page="/unprotected/loginLevel3.jsp"/>
</auth-config>
```

5.5 Mapping views

For each presentational class that is part of the frame split, a JSP view-page is created. The presentational classes are also used to define areas where CSS is applied. This is done in the next sub chapter.

JSP view-pages:

For the example system we get:

Header View -> header.jsp
Menu View -> menu.jsp
Balance View -> balance.jsp
Consult Transaction List View -> transactionList.jsp
Transfer View -> transfer.jsp
Payment View -> balance.jsp

Each of these pages follow a template page structure. The differences are described by the presentational classes on each page along with the presentational symbols. These symbols are standard symbols and can be put in a form of a library so the graphical layout designer only has to think about the placement and the description of the style sheet that is applied to each element.

5.6 Adding CSS

CSS can be applied to documents in three different ways.

1. By linking a CSS-file in the top of the HTML document – normally inside the header

```
<link rel="STYLESHEET" type="text/css" href="stylesheet.css">
```

2. By embedding CSS on the page using a <style> tag

```
<style>...</style>
```

3. By using inline style sheet on each element.

```
<img src=”pic.gif” style=”...”>
```

The most efficient way for a home banking web site (and most sites in general) is to use the first method where a style sheet file is loaded. In this way all the styles are kept outside the view pages and reusability in design layout can be maximized. It is not unlikely that the graphical layout designer suddenly wants to make global change that need to effect all pages.

Imagine for example that the background color for each page is described by a tag in the stylesheet.css file. When the designer wants to change the background color on all pages he would only have to change in one place.

In the stylesheet.css file there would be a tag like:

```
.cssBackground {
  background-color: #f0f0f0;
}
```

All pages with the with the following body tag would have the color: #f0f0f0 (RGB-value) as the background color.

```
<body class=”cssBackground”>
```

The CSS styles are added for each CSS-element defined in chapter 4.5.3

For the example system we get:

```
.cssHeader {
  background-color: #9CB5C8;
  margin-left: 20px;
}

. cssMenu {
  background-color:#8AA4B8;
  margin-left: 0px;
}

. cssMain {
  background-color:#9CB5C8;
  margin-left: 20px;
}
```

```
. cssPageName {
  font-family: Verdana, Arial, Helvetica, sans-serif;
  padding-top: 12px;
  padding-bottom: 8px;
  padding-left: 20px;
  text-align: left;
  font-weight: bold;
  font-size: 10pt;
  background-color: #8EA7BA;
  width: 500px;
  height: 43px;
  vertical-align: center;
  border-bottom: 1px solid #8198AA;
}

. cssPageNameText {
  font-family: Verdana, Arial, Helvetica, sans-serif;
  color: #336688;
  padding-top: 0px;
  padding-right: 4px;
  text-align: right;
  font-weight: bold;
  font-size: 8pt;
}

. cssMessage {
   padding-left:40px;
}
```

There is defined two CSS styles for this text because it is a result of an operation that should result in two different views:

```
. cssMessageTextOk {
  font-family: Verdana, Arial, Helvetica, sans-serif;
  color: #2266ff;             // blue color – signifies a success full result from an operation
  font-size: 9pt;
  font-weight: bold;
}

. cssMessageTextError {
  font-family: Verdana, Arial, Helvetica, sans-serif;
  color: #ff6622;             // red color – signifies an error happened during execution
  font-size: 9pt;
  font-weight: bold;
}

. cssButton {
  font-family: Verdana, Arial, Helvetica, sans-serif;
  color: #336688;
  font-weight: bold;
  font-size: 8pt;
}




. cssSubMenu {
  padding-top: 2px;
  padding-bottom: 2px;
  text-align: left;
  padding-left: 3px;
  font-weight: bold;
  font-size: 8pt;
  font-family: Verdana, Arial, Helvetica, sans-serif;
}

. cssAnchor:link {
  color: #0033cc;
  text-decoration:none;
}

. cssAnchor:hover {
  color: #3366cc;
  text-decoration:none;
}

. cssAnchor:visited {
  color: #000033;
  text-decoration:none;
}

. cssAnchor:active {
  color: #3366cc;
  text-decoration:none;
}
```

In the appendix there is an example of the full style sheet file: *styles.css*

5.7 Implementing the Authentication Model and HTTPS

This subchapter describes the implementation of the authentication model and the steps needed to create HTTPS connection.

Authentication Model Implementation

The authentication implementation consists of a class file (BankingHook) with three functions and the auth-config.xml file mentioned in chapter 5.4.

When the client accesses the home banking application for the first time a session is created in the web server. This session is valid until it times out (web server configuration) or until it is being invalidated either by the application or when the browser window is closed by the user. The session contains the client id that identifies the client and the authentication level that the client is authenticated to for the current session.

In the following the main functionalities in BankingHook is explained:

logIn - tries to login to the given authentication level - this is done by accessing the local homebank database and verifying the user id and password for the given authentication level

logOut – invalidates the session and thereby "forgets" the client id and authentication level and then redirects to the welcome page

securityCheck – checks the authentication level for the command – this is done by getting the authentication level from the session and checking against the authentication level for the command supplied by the Controller Servlet.

HTTPS implementation

To implement HTTPS in the home banking application a certificate for the server is needed. This certificate is used by the web server to identify itself. It contains some data like name of holder, expire date etc., the signature of the certification authority and the public key used for encryption.

To implement the HTTPS connection on the iPlanet web server the following steps were taken:

- A trust database for certificates was created
- A certificate was requested
- The certificate was installed on the server
- In the last step the encryption was turned on

Requesting a certificate can be done in two ways:

1. Either a certificate is requested from one of the big and trusted certification authorities like VeriSign[26]. This has the advantage that all the most known browsers already trust VeriSign.
2. The other way to request a certificate is by installing your own certification authority. In this thesis the iPlanet Certificate Management System was installed and used to create a certificate.

Once the encryption is turned on with the web server the only way to access the home banking application is by using an HTTPS connection.

Finally in the home banking application itself it was necessary to modify the code where the forward to the JSP page is made in the Controller Servlet.

The important parts of the code that does the forward is shown below. What is important to notice is the urlString where the "https" is used as part of the url. The *forwardPath* contains the URL to be forwarded to. Furthermore the main thing to create the forward with HTTPS is the HttpsURLConnection which sets up an HTTPS connection. This connection is used to connect an input and an output stream through which the forwarding of the bean containing the functionality is done (the purpose of the bean forwarding was described in chapter 5.3).

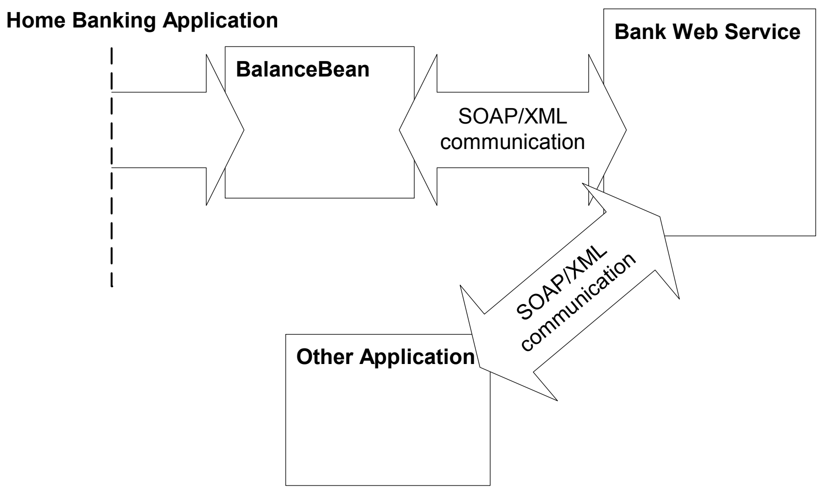
```
String urlString = "https://mistral"+forwardPath;

URL url = new URL(urlString);
System.out.println("Connecting to " + url + "...");
HttpsURLConnection con = (HttpsURLConnection)url.openConnection();

con.setRequestMethod("POST");
con.setDoInput(true);
con.setDoOutput(true);
```

```
OutputStream outStream = con.getOutputStream();

ObjectOutputStream out = new ObjectOutputStream(outStream);
out.writeObject(b);
out.flush();
out.close();

InputStream stream = null;

if (con.getResponseCode() >= 300) {
    System.err.println("Received Error: "+con.getResponseMessage());
} else {
   stream = con.getInputStream();
}

OutputStream os = response.getOutputStream();

for (int c = 0; (c=stream.read()) != -1; ) {
    os.write(c);
}
os.flush();
os.close();
```

5.8 Bank Simulator – Glue Web Service

In this subchapter the bank simulator implementation is described. The bank simulator implementation serves as a proof-of-concept to test the home banking application and is not intended to represent a real bank legacy system.

To illustrate and amplify the separation of the systems in the home banking application, I decided to use a different technology for the implementation of my bank simulator than the one used for the home banking application itself. The Glue Web Service[20] is a free software product (shareware) that provides easy installation and configuration of a web service. It supports Java Technology which gives me the possibility of invoking a Java-method from the home banking application. The communication is done with SOAP/XML and Glue generates the WSDL code used in the communication. By using the SOAP/XML format in this way the technology and implementation in the bank is effectively separated from that of the home banking application. This means that the bank system can be replaced and changed without the need of modifications to the home banking application. As long as it provides the same web service with the standard formats SOAP/XML.



Sharing of Web Service

The bank simulator has the following directory structure inside the Glue directory:

```
  src
   |
 banking
```

The bank simulator consists of the following classes:

BankDatabase, BankOperations, IBankOperations and Pub. Following is a description of each of the classes along with code parts that illustrate the functionality of the bank operations as a web service.

BankDatabase

This class sets up the bank database and contains functions to execute SQL selects and updates. These functions are called from the BankOperations class. The following functions exist:

BankDatabase - connects to the database with the driver

createDB - runs the scripts that create the tables and values in the database

execDB - executes a *statement.execute()* that reads from the database

execUpdateDB - executes a *statement.executeUpdate()* that writes to the database

BankOperations

This class contains all the bank operations. These operations use the BankDatabase to execute the SQL statements that implements the bank operations. Each of the functions in the BankDatabase related to the example system is described below.

All of these functions work on the bank database and retrieve or stores their input and output data with SQL statements *select* and *update.*

getAccounts - gets the accounts for the client and returns them as string array

getBalance - retrieves the balance for a given account and returns it as a string

getTransactionList - retrieves the a list of transactions for a given account and returns them as a string array

execTransfer - executes a transfer from one account to another by updating both the source account balance, the destination account balance, the transfer table and the transaction table

execPayment - executes a payment by updating the account with the payment, the payment table and the transaction table

getPaymentTypes - is used to show the payment types on the payment page and also part of the description in the transaction table

IbankOperations

This is an interface class that is used by the tools provided by Glue to create descriptions in the WSDL file. Following is an example of a function in the bank and the generated WSDL file. This file is typically used by the different companies that develop applications using bank operations. To go into details about the WSDL file is outside the scope of this thesis, but this example illustrates the easy way to distribute a web service.

Code for the Show Balance operation:

```
 ...
/**
 * Returns the balance of an account
 * @param account The account number
 * @return The balance
 */
String getBalance(String account);
 ...
```

Resulting part in the WSDL file:

```
 ...
- <message name="getBalance1In">
- <part name="account" type="xsd:string">
  <documentation>The account number</documentation>
  </part>
  </message>
- <message name="getBalance1Out">
- <part name="Result" type="xsd:string">
  <documentation>The balance</documentation>
  </part>
  </message>
...
- <operation name="getBalance" parameterOrder="account">
  <documentation>Returns the balance of an account</documentation>
  <input name="getBalance1In" message="tns:getBalance1In" />
  <output name="getBalance1Out" message="tns:getBalance1Out" />
  </operation>
 ...
```

Any application developer that needs to interact with the bank only needs access to the WSDL describing the web service. He doesn’t need to care about the system that implements the service. Furthermore if the bank at a later state decides to change or upgrade their system, they can do so without problems as long as they make sure the same web service is provided.

Pub

This class publishes the bank operations as a web service on port 8004.

```
package banking;

import electric.registry.Registry;
import electric.server.http.HTTP;

public class Pub {
  public static void main(String[] args) throws Exception {

    // start a web server on port 8004, accept messages via /glue
    HTTP.startup( "http://localhost:8004/glue" );

    // publish an instance of Exchange
    Registry.publish("bankoperations",new BankOperations());
  }
}
```

When running the Pub class the web service is available on this port:

```
http://localhost:8004/glue/bankoperations.wsdl
```

(Requesting this page on the home banking application computer makes the browser show the WSDL file that describes the web service. The complete file is shown in the appendix)

to connect to this web service it has to be invoked. An example of this is shown here:

```
String wurl = "http://localhost:8004/glue/bankoperations.wsdl";
String iMethod = "getBalance";
String balance = null;
String chosenAccount = "0";

try {
        balance = (String) Registry.invoke(wurl, iMethod, new Object[]{chosenAccount});
    } catch(Throwable t) {
        System.out.println("Throwable exception trying to invoke method: " + iMethod);
        t.printStackTrace(System.out);
    }
```

5.9 Problems and considerations during implementation

Building the authentication model and access restrictions proved to be a hard task. In the following is described some of the problems and their solutions that were encountered during the implementation process.

Basically the access restrictions was done by protecting all the important files from access from any other machine than the machine with the web server and the web application. Thus assuming that no one who is not allowed has access to the machine itself. Access to the protected files has to go through the ControllerServlet which contains the authentication protocol. The identifier for the machine is the IP-address.
One of the first problems with creating my own authentication and access control was to configure the web server to grant permission to files on the machine only to the machine itself. This couldn't immediately be done with the Tomcat web server[21], which was the first choice of implementation. Therefore I switched to using the Netscape iPlanet web server. Here you can easily configure the permissions so that some files are only accessible from a certain IP address – the one of the web server machine.
By using the iPlanet web server this works fine, but during implementation one significant problem occurred. When doing the forward from the ControllerServlet to the protected files, the forward uses the request from the client machine. The web server would therefore not allow access to the files, since it thinks it is a client trying to access the protected files.
To make it possible for the ControllerServlet to access the files and still keeping them protected from outside access a new request that doesn't have the requested IP-address of the Client machine has to be generated. This can be done by creating a post from inside the ControllerServlet*). This solved the problem of protecting the files from anyone but the ControllerServlet, but it generated another problem with fitting the first proposed MVC-model. In this model the session on the client side is used to hold the class instantiated by the ControllerServlet and used by the view (JSP) pages. But when creating a new request a new session for that request is also created and the server "pretends" to be the client with that request. For obvious security reasons it is not possible to change the request nor change the session on client side. Therefore the only way to make the server side instantiated beans available to the JSP pages is by transferring them in some way.
One last thing that had to be dealt with when creating a new request was that a cookie appeared on the web server machine for each request from the client browser. This is because when we create a new request a new session is created and all session handling in JSP pages are handled with the use of cookies. Since this cookie serves no purpose for our model a configuration is made to prevent this cookie from popping up (and avoid hiring a guy to accept or reject every cookie).

At a later state the HTTPS was implemented and the forwarding in the Controller Servlet had to be changed to support this. The HTTPS implementation was described in chapter 5.6 which shows the final implementation. Before reaching this point the Java API had to be expanded with two class libraries**) along with their patches. This shows a typical real life development process where the flexibility of Java comes in to hand. When the standard JDK API doesn't provide functionality enough for your task it is easily expandable.

*) For the implementation of the post function an HTTPClient package[14] of classes were added to the application. This package provides an HTTP client library including request methods HEAD, GET, POST and PUT.

**) iSaSiLk 3 containing HTTPS connection support and IAIK-JCE 3.0 containing an encryption package

7. Documentation of a home banking site

In this chapter I introduce a method to document a home banking site. It is split up into two parts. One describing the functionality and another describing a simple navigational structure that can be handy to include on a web site.

7.1 Documentation of each web page

This part describes each web page in the home banking application and the function it implements.

From this documentation it should be possible to build the navigation of the web site as well as understand the functionality described on each page. Furthermore the layout of the page is described in regards to use of style sheet.

Each page contains the following parameters:

Name – the web page name
Description – a short description of the functionality of the page
Frameset – the frameset the page belongs to
Function - which function it implements
  Group - which group of functionalities it belongs to
  Level - the authentication level for each function
Elements – the visual elements on the page
  CSS – each style used for the elements
Input – the parameters sent with the page
Buttons – tells whether the page has an execute and clear button
Links – links on the page

Each frameset is defined with the following parameters:

Description – a short description of the functionality of the page
Frames – a list of the frames and their frame name that are shown by the frameset

As an example the balance page is shown:

Name	balance.jsp
Description	shows the balance for a given account
Frameset	index.jsp
Function **Group** **Level**	getBalance consultation 1
Elements **CSS**	page name, page name text, account cssPageName, cssPageNameText, cssMessage, cssMessageText, cssButton
Input	account number
Buttons	
Links	

The top frameset is shown as an example:

Description	shows the balance for a given account
Frames	header.jsp - headerFrame menu.jsp - menuFrame main.jsp - mainFrame

7.2 Web Site Map

This part describes a way to show the organization of the web pages on a web site. This is generally called a web site map.

A web site map can be created in several different ways depending on the purpose and the target group. The purpose of a web site map on a web site is to show visitors the structure of the site as well as give them a possibility of going straight to a page. It has to give an impression of the layout of web pages available and the easiest way to get to them. A good way to do this is to group things and use menus, possibly with lower menus.

For the example system a simple grouping of pages according to subject provides the user with a simple and easy overview of the pages. The web site map also provides a way for the user to go directly to a page. This example shows the grouping of pages according to functionality, but other criteria may be followed. All the pages could for example be listed alphabetically.

Consulting functions:
- Balance
- Transactions

Operation functions:
- Transfer
- Payment

Personalization functions:
- accountNewName

8. Future expansions

In this chapter I give some suggestions to improve the home banking site application. These suggestions consider both the functionality, separation of concerns and security enhancements.

8.1 Improving Application Functionality

Improving functionality stability

An important general functionality to implement in a real home banking application is to check for a duplicate operation. If a user by accident tries to execute the same operation twice the system should give a warning message. For example a user trying to do the same transfer twice. This is a typical error among home banking clients.

Improving database stability

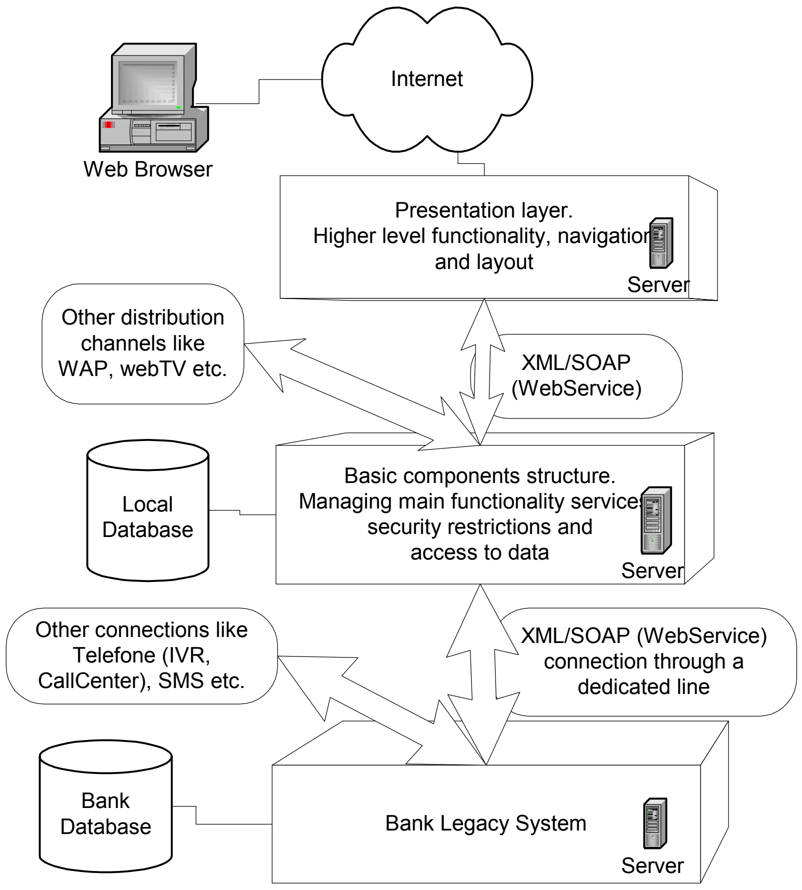
If the database suffers a breakdown in the middle of a bank operation that possible requires multiple database updates. The system should be able to check and recover from this. Either by continuing where it left of or go back to the start of the operation and make a *roll back* in the database.

8.2 Administration Application

An obvious expansion to the home banking application is to create an administration application. The main purpose of this application would be to administrate clients and accounts. With operations like adding, deleting clients, administrate the authentication levels for the client, etc., for the home banking application service.

8.3 Expanding model to support multi-channels using the Web Service format

The functionality at the middle layer can be reused by several channels, like a home banking site (through a browser), WAP (with a cellular phone), TV (using a Set-Top box), etc..
At the top level the navigation, layout (graphics) and functionality specific to the channel can be defined. By inserting an extra level in the system architecture as shown on the figure below a better way of reusing code and maintenance of a large application environment is improved. The Web Service format can again be used as a good way of separating the implementation environment.



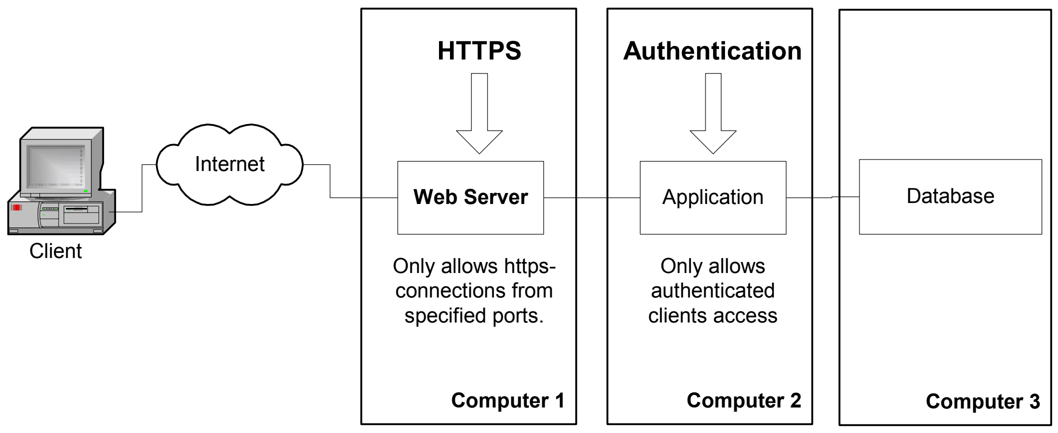
8.4 Improving separation at top level in the model

When implementing view (JSP) pages in the current model the web designer has to know the tag-based language HTML (and maybe also some JavaScript to implement dynamic HTML). To get the content (created by the functionality) to be presented on the page he has to use JSP scriplets. These scriplets call java functions in the Java Beans representing the functionality. One way of avoiding this way of mixing the tag-based HTML with the Java-based scriplets would be to implement a tag library where each function call is represented by a tag. With this implementation the whole web page is described with tags and the web designer only has to know how to use and which tags can be used on which pages. This both helps the web designer implement the page but also gives the possibility of running programs that verifies (tag-validates) the syntax of the page. Thereby making it easier to implement error-free pages.

8.5 Improving security in the implementation

In the current implementation architecture the whole application with web server and database is implemented on the same computer. This means that gaining access to the computer means gaining access to the protected files and the database. All access to the computer goes through iPlanet, that supposedly only gives access to the port where the web server is running. The web server then controls the access to protected and unprotected files and accesses the database. iPlanet is used throughout the world as a web server and is

therefore also a target of attack. Some holes in iPlanet has been found and several patches exist to solve this.
A way to improve this architecture that needs less reliance on iPlanet to protect the files could be to separate the iPlanet control from the web application and the database. Basically by using three computers. The first computer contains the iPlanet control, the second computer only accepts requests from the first computer and contains the web application and the third computer contains the database that can only be accessed from the second computer. Several filters can be added between the computers depending on the functionality of the system. For example if the home banking application is only available in a certain time period, one could add a filter between the second computer and the third computer that only grants access in that certain time period. The rest of the time there is no access to the database. This architecture also improves the availability of the database and the web application. If someone makes the first computer go down the others can continue to work and provide their service possibly to other computers.



8.5 Implementing PKI-security

In the current security model for the home banking application only the web application is authenticated. This means that the home banking application trusts the user if he can present a valid user id and password. A more secure situation would be if the client also had to be authenticated to access the application. This can be achieved with the use of PKI - Public Key Infrastructure[4] which implements mutual authentication.

PKI provides a full set of security services with the purpose of preventing security threats.

There exist four parts in a PKI.

- CA (Certificate Authority) which are organisations that issue digital certificates. This is normally a widely trusted party.
- An Authentication Service
- Services
- Business users in a client server environment.

A typical scenario of using the PKI

- First the new user requests the CA for a certificate
- The CA responds with the certificate for the user

- With the certificate in hand the user accesses the application and presents the certificate
- The application checks with the authentication service

Implementing a PKI infrastructure will provide a very safe environment, but the disadvantages are that it is expensive to implement and the communication will be slower.

9. Conclusion

In this project a home banking application and the surrounding environment has been investigated. Furthermore an implementation of the complete system has been made. Following now is a discussion of the parts of the project where the most important experiences were made.

9.1 Modelling a web site

A method was developed to automate the design process of a home banking application. This model separates the concerns of web sites in layout, interactivity and functionality.

By separating the concerns in this manner several important goals are achieved:

- reusability in each part can be applied
- the layout designer can specialize on layout, the navigational expert can focus on navigation and the programmer can concentrate on the functionality

The reusability was implemented in each part of the web site.

Layout – by splitting up the content on the page in blocks called views – groups of style sheet elements can be reused.

Navigation – the navigation was described in groups of similar functionality and these groups can be reused.

Functionality – in this thesis an example of reusability was given. It was recognized that most banking operations depend on a selected account and therefore this module was reused.

Whenever possible the steps in the model are described in an automated way opening the door for automated generation. Tools can be build that ease the task of implementing and updating the web site.

In the navigational design and layout design a UML-based method is described. Using UML as a modelling language has the advantage of using a well-known standard thus making it possible to use already defined case tools in the development process.

Applications with similar structure

The closest application to a home banking application is a corporate banking application. Most of the structure in the form of navigation, layout and parts of the functionality would be directly reusable. A typical scenario is for a bank to have both applications and implement them with similar type of navigation, possibly some different layout parts and colours and a more extended functionality.

In this thesis a home banking application contacting a bank is shown. A lot of other types of applications could implement the same structure. It would be practical to have online stores, stock market sites, etc. applications that can benefit from direct contact with one or several banks. This would make payments easier since they can be done in directly and immediately in your bank.

9.2 Security in a home banking application

It was furthermore investigated what security concerns are necessary and how they fit into the model. In the security investigation a custom made authentication model was build that gives the possibility of applying your own authentication/access policy.

Basically two security services are directly implemented in the model. The first is authenticity where the authentication level for each function had to be defined. The other is the audit trail security service where each operation done by the user is logged for later documentation proof. To some extend the availability was also implemented by making sure that the bank would only be contacted if the homebank database was running and the operations therefore could be logged.

From the experiences made during the implementation of the security model I think it would be better to separate the security implementation completely from the implementation itself. That would make it easier to focus on each part separately and thereby improving the actual implementation. It would also make it easier to reuse the security part for a different web site where security plays a similar important role, but where functionality is quite different.

9.3 Web Service as a communication platform

It was shown how Web Services can be used as an efficient way of communication. By using the Web Service format, the two units, in this case the bank system and the home banking application, can be seen as two individual and independent systems. This facilitates maintenance of the systems and gives more freedom in the choice of implementation technology.

The complete system involves setting up a web server and database server for the home banking application, implementing a bank simulator with another database as a web service.

Using the Glue Web Server proved to be very easy to configure and use. The way Java functions can be directly invoked makes the web service seem transparent and it gives a good platform for real life applications. Not much thought on how the communication works is needed. This makes it a good tool when one of two communicating tools need to be changed or updated. As shown in this thesis a simulator can be set up during implementation where everything is tested. Once everything is working correctly the application is connected to the real application – in this thesis it would be to the bank legacy system.

In the future more and more applications will be linked together most likely using the web service format.

9.4 Documentation of a home banking application

Part of the process of modelling a home banking application is defining ways of describing the functionality, the navigation and the layout in a formalized way.

In this thesis the modelling was first build and the documentation came afterwards. I think it would have been a better idea to it in the other way around. Either by making the documentation first and then the modelling or try and make them at the same time. By doing the documentation would give a different angle as to what the home banking application should contain.

9.5 MVC as implementation platform

The MVC – Model View Controller defines a model for implementing dynamically generated web sites. It uses the technologies from Java to separate the concerns. With a combination of JavaBeans (Model), Java Servlets (Controller) and JavaServer Pages (View) it provides the platform for the home banking application implemented in this thesis.

Generally the model has a good structure for implementing the model for building a home banking application described in this thesis, but is complicated to set up and debug. In this thesis a Theseus implementation was used as the basis and customized to implement the home banking application. I found it to support the web site modelling well, but giving some restrictions on flexibility. It didn’t support my authentication model immediately and a lot of work was done to solve this problem.

If I was to redo the project now I would create my own MVC implementation and thereby be able to fit the home banking modelling better with the actual implementation.

References

[1] *Andreas Heberle, Jörn Rehse, Bernd Ornasch and Börje Sieling.*: Utilizing Abstract WebEngineering Concepts: an Architecture. IEEE MultiMedia, 0-7695-0981, September 2001.
[2] *William Stallings*.: Network Security Essentials - Applications and Standards, Prentice Hall, 1999, ISBN 0-13-016093-8.
[3] Niels Bach, En metode til udvikling af web-baserede systemer. 2001.
[4] PKI description homepage, http://www.id2tech.com/topmenu/smartcard/pki.asp.
[5] JavaBeans homepage, http://java.sun.com/products/javabeans/.
[6] Java Servlet homepage, http://java.sun.com/products/servlet/.
[7] JavaServer Pages homepage, http://java.sun.com/products/jsp/.
[8] MySQL homepage, http://www.mysql.com/.
[9] *Daniel Schwabe and Gustavo Rossi*.: The Object-Oriented Hypermedia Design Model (OOHDM).
[10] *Nora Koch, Hubert Baumeister, Rolf Hennicker and Luis Mandel*: Extending UML to Model Navigation and Presentation in Web Applications.
[11] *Rolf Hennicker and Nora Koch*: A UML-based methodology for Hypermedia Design.
[12] Caixa Geral do Depositos home banking site, http://www.cgd.pt/particulares/produtos_servicos/servicos/servico_caixadirecta_online/demo/.
[13] iPlanet web server home page, http://www.iplanet.com/.
[14] HTTPClient home page, http://www.innovation.ch/java/HTTPClient/.
[15] SOAP home page, http://www.xml101.com/news/news_soap.asp.
[16] WSDL home page, http://www.learnxmlws.com/tutors/wsdl/wsdl.aspx.
[17] Theseus MVC-model home page, http://www.brainopolis.com/theseus/index.html.
[18] XML home page, http://www.xml101.com/xml/default.asp.
[19] CGI home page, http://www.cgi101.com/class/.
[20] Electric GLUE web service home page, http://www.themindelectric.com/.
[21] Jakarta web service home page, http://jakarta.apache.org/tomcat/.
[22] Macromedia Flash home page, http://www.macromedia.com/software/flash/.
[23] CSS tutorial home page, http://www.intranetjournal.com/articles/200101/csstutorial1a.html.
[24] JDBC database interface home page, http://java.sun.com/products/jdbc/.
[25] Smart Card home page, http://www.smartcardbasics.com/.
[26] VeriSign home page, http://www.verisign.com/.