TAKING A JEE APPLICATION OFFLINE:
A Comparative Case-Study

Student: Alexander Bij
Mentor: Tijs van der Storm

School: University of Amsterdam
Master of Science 2007/2008

Abstract:

The browser is widely used as a thin-client for web-applications. It works only if both client and server are connected. New techniques enhance the functionality of the browser to let a web-application operate when the client is not connected to the server. In this thesis we investigate which techniques are able to do so. After the investigation we migrate a JEE web-application with two of those techniques and add offline functionality. A web-application with a small domain is created as the baseline for the research. During the migrations of the baseline we try to reuse the original source-code. We measure the changes and compare the results of the migrations. With the different techniques used during the migrations and supporting frameworks its different to point out the one that is most suitable for our case.

Contents

1 Introduction

The thesis is about a java enterprise edition [JEE] web-application that undergoes two migrations where offline functionality is added. Traditional JEE web-applications are applications with a thin-client, the browser, sending requests and receiving responses from the server over the HTTP-protocol. The server contains all the business logic and based on the received requests it generates HTML. The HTML page is sent back to the client. This works fine in most cases, but there are times there is no connection to the server. The migration of the JEE web-application enables a part of the existing functionality to work offline. The client should be able to perform a task without being connected to the server.

The JEE web-application is the baseline for two migrations. Two techniques are chosen during the investigation enabling offline storage that is applied to the baseline so that no connection is needed. The required modifications and additions to the baseline are measured to give a result how the technique supports reusability of the JEE web-application. The goals of the research is to point out what technique can be used to support the migration with high reusability of the baseline.

1.1 Motivation

Before we describe the motivation, lets explain what is mend by offline functionality: the user on the client can perform a tasks without being connected to the server. The motivation for the addition of offline functionality to a web-application is:

Improve the user experience: The connection to the server and the data transfer delays the load time of web-pages. The client (browser) waits until the web-page is loaded before it can continue his task. Enabling offline functionality requires that the data is available on the client. The data is stored locally and has fast access. In a traditional JEE web-application all data have to be sent each time the client requests a page. The fast access of data results in faster load times and thereby improving the user experience.

Ubiquitous, continue work while being disconnected: The client is no longer required to have a connection to the server and can still perform his tasks. The client can work on places where the internet is not available or suddenly got disconnected from a connection, without losing data.

Reduce the server traffic load with client-side storage: Unchanged datasets that are requested by the client are normally sent again. This is not necessary with client-side data storage, because the client has the unchanged data available locally. The server can only send updates, which reduces the size of the response. This reduces the amount of traffic generated by the server and the smaller dataset can reduce the time gathering the data.

Use the advantages of the web: Traditional JEE web-applications can distributed easily. The client can access the application with a web-browser and new version of the web-application does not require any user actions. 'Fat-clients' or desktop applications have to persist the application updates to the clients is a more verbose way. During the migration we want to keep the distribution the same way as traditional web-applications.

1.2 Research Question

The research question is: *"How to migration a JEE web-application to add offline functionality with reuse?"*. During this research I want to learn which techniques are currently capable of doing this and how to apply these techniques to a existing web-application. Besides the technical point I want to learn how to execute a research and document it readable for other viewers. During the education I have learned how to write an essay, find information in papers and I want to apply this knowledge in the thesis.

The techniques which are able to add offline functionality are selected during the investigation. The investigation is executed with new techniques that enrich the traditional web-applications and offer client-side storage. Two techniques are selected which are used in the case. In the case are the two techniques applied to the same traditional web-application, the baseline, to enable the offline functionalities. Offline functionality is defined in this context as being able to complete a task within the web-application on the client without a connection to the server. This can mean that the server is 'down' or the network has a problem or the client is working in offline mode.

The research is about possibility of making functionality available offline. To compare these techniques we look at the reuse of the traditional web-application. Sogyo, is the company where this thesis is created, is interested in what modifications are required to add offline functionality to a traditional web-application. The following question is created with reusability as a quality attribute created by Sogyo:

- How much can be reused of the original web-application if offline functionality is added?

The size of the new functionality is measured to see the amount of LOC (lines of code) required to create the offline functionality and how it relates to the original source of the web-application. The information indicates the change of the original source-code by the techniques. With the differences in source-code (addition, modification and deletion) we can indicate the reuse of the baseline. The web-applications' architecture exists of separated components and the migration with the techniques will have impact on the architecture. Besides the lines of code, we look at the impact of the architecture.

1.3 Experiment

The initial situation a JEE web-application is created that serves as the baseline to test the techniques. The web-application is a Java application with a small domain for giving a valid result so it can be completed within the given time. Sogyo does not have an appropriate JEE web-application that can be used as baseline for this case. A web-application is created for the case study, called JDigiNotes.

We added offline support to JDigiNotes using two techniques. Selected functionalities of the baseline are created in an offline variant with the chosen techniques. Both techniques implement the same functionalities so that the differences in implementations are measurable. The goal of the measurement is to point out which technique, that supports high reusability of the JEE web-application, can be used best to add the offline functionalities. The amount of source-code of the baseline is measured and the source-code is measured after the migrations. The changed source-code in the applications indicates which technique can be used best for high reusability. Besides the source-code the change in architecture is discussed. The original web-application is the baseline for the experiment.

To test the reusability of the original web-application for the selected techniques, the size is measured and the findings are discussed. The size of the baseline is measured in LOC (Lines of Code) to have an indication what the size of the project is. The size in LOC is also measured after the migration. The LOC metric visualizes the amount of changed code of the baseline application. The size of additional code for the offline functionalities and the size of the changed code of the original baseline are measured. The more LOC are changed or added implies a bigger modification and the more effort it costs to migrate the web-application.

The reuse can be indicated by the LOC. We want to look at the change in architecture of the migrations. The migrations with the techniques could introduce a new programming language, configuration settings and new components. We try to point out which components of the baseline can be reused and what changes are made. The changes at architectural level are presented in the results and discussed in the comparison.

To have a valid research we have setup requirements that the research should imply. These requirements are followed during the execution of the migrations. In the chapter experiment is explained how these requirements are followed. The requirements for the experiment are as followed:

- *Independence:* the web-application functioning as the baseline should be independent from the techniques. The creation of the baseline should not be influenced by the chosen techniques.
- *Repeatable:* the experiment is executed with new techniques and some of them are not in the final release state yet. New techniques offering offline storage will appear over time and others can be changed with new features. The experiment should be repeatable with other offline implementations.
- *Unchanged Baseline:* the experiment is executed on an existing JEE web-application functioning as the baseline. The baseline is a server side Java application operating in a ServletContainer[1] [ServletContainer]. This setup should not be changed during the migration to add offline functionality. The code of the baseline may be changed but the server setting cannot.
- *Measurable:* the result of the migration should be measurable to be able to compare the techniques to give a valid conclusion. In this case the LOC of the baseline are measured and the LOC of the web-applications after the migration with the techniques. The changed and added LOC to the baseline indicate what parts of the baseline are be reused and what parts need to be changed.

[1] A Servlet container is a specialized web server that supports Servlet execution. It combines the basic functionality of a web server with certain Java/Servlet specific optimizations and extensions.

The first step in the research is to have a clear view of the problem and how it can be solved. The problem of traditional web-applications and the promising techniques trying to solve them are described in the next chapter Background.

After investigating the techniques we start the Experiment and define the scope and create the baseline. Despite Sogyo did not have an appropriate web-application, we created JDigiNotes. In the chapter is described how JDigiNotes is build and validated whether it is representative for this research. The migrations with Gears and AIR are described afterwards.

The outcomes of the measurements are presented in the chapter Results. Here we explain what the numbers mean that we measured after the migrations. These are discussed in the Comparison. The results of the migrations are put together in a table and looked at the differences.

The threats to the research are described in Threats to validity. We end this thesis with the Conclusion of this research.

1.4 Case Study

The case study is done with a JEE web-application that is the baseline for the experiment. The baseline application is called JDigiNotes. The 'J' refers to Java, the programming language where the application is created with and the JDigiNotes to Digital Notes.

The JDigiNotes is a digital version of the yellow posts-its. Notes are created by one person and can be seen by others. The notes can be added, changed and deleted by persons who are authorized and authenticated. In the digital version the person who is the creator of the note is the owner and is the only one who can delete or change the note. Authenticated and unauthenticated users can see the notes. It is a small domain that allows the offline functionality to be added within the given time and makes it easier to measure the results. The functionalities are displayed in the use-case diagram Figure 1.

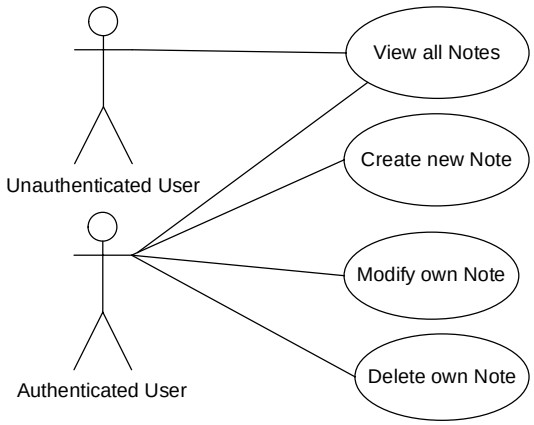

Figure 1: Functionalities of JDigiNotes in UML Usecase diagram.

During the experiment a part of the functionalities of the baseline are made available offline. First the techniques able to support this migration are selected and two of the techniques are chosen.

We have chosen for JDigiNotes, because it is an easy to understand domain and contains a small number of functionalities that can be found in traditional web-applications. For example online shops or online agenda/email are examples where domain objects undergo the actions of figure 1. The architecture of JDigiNotes is similar to the architecture used in the Sogyo Bankcase and other java projects, but with more functionality and are thereby more complex. The BankCase is used to educate junior software engineers how to create a java web-application with best practices and following Sun guidelines for web-applications.

Sogyo is interested in new techniques and their capabilities. The research is created based on the term ubiquitous software. The term that is mend for software that operates online and offline, without interrupting the user's actions. The research has become a migration of a web-application to make a part of it available offline. The BankCase was the first chosen, but could not be used for the research. A complete version of the BankCase was only available in the .NET-programming language. The created Java implementations had to many flaws, shortcomings or could not be found in the repository. An option was to create the BankCase, but that would take too much time.

2 Background

2.1 Limitations of Traditional web-applications

In this section traditional web-applications and their limitations are explained for the investigation. A traditional web-application uses HTML for the web-pages to communicate with the web-browser over the HTTP-protocol. The presentation of the application is described in a client-side language (commonly HTML-4 with CSS) and the business logic is described only on the server with a server-side language (ASP, JSP, PHP etc.). The traditional web-applications are not able to support offline functionality.

The client sends a request to the server via the HTTP protocol and the server dynamically generates the HTML page with the server-side language. The response is sent back to the client. [Christodoulou 1998] and [Duhl 2003] point out the four major limitations of traditional web-applications:

1. Process; multiple screens are required to perform a single task.
2. Data exploration; the traditional web applications do not support interactive data exploration and manipulation. Users have to search and navigate through various static pages to view the desired data.
3. Configuration; many web applications require the configuration of a product or system from multicriteria choices and are unable to present the customized product or system to the user in an intuitive way and in a single step.
4. Feedback; all the actions from the client will result in a page refreshment. The user can't stay on the same page to complete an action.

Traditional web-applications cannot support offline functionality and have more shortcomings. RIA can be used to overcome these limitations. These are described in the next chapter.

2.2 Rich Internet Applications

Rich Internet Application is a new type of Web user experience that is more interactive, more responsive, and more engaging than the web applications with traditional HTML web interaction [ThroughLeadership]. The Macromedia point of view about is that RIA's combine the best user interface functionality of desktop software applications with the broad reach and low-cost of an application providing a more intuitive, responsive and effective user experience [Duhl 2003]. RIA are not meant to replace traditional web-applications. The concept of richness in RIA extends the traditional web in three aspects: data, presentations and communication capacities [Morales 2007]. The data can be explored in a more intuitive way. The presentation is the look & feel of the RIA-application, which is more flexible than HTML-4. The communication can be a-synchronous and does not have to disturb the users tasks.

A RIA can be described as *a web-application with an enriched and more responsive user interface*. The web-application term refers to distribution of the application over the internet. The initial installation or startup of the RIA can be reached from the client web-browser over the internet; new versions can be distributed more easily than fat-clients.

RIA's *can* run in the context of the web-browser, but *don't have to* run in the context of the web-browser. For example Adobe AIR applications do not run in the browser, but run in the Adobe Integrated Runtime. The term 'enriched' means more opportunities for animations, shapes and media than a traditional web-application.

Traditional web-applications need a refresh of the page after a click on a link or button, users have to wait for the response of the server before they can continue their task. RIA's overcome this limitation with asynchronous communication techniques like AJAX, which allows the user interface to be uninterrupted and the user can proceed performing his tasks while the function call is processed in the background. An application can be labeled RIA if all these features are available. (Operate in a web-browser kind of runtime, presentation in Rich UI, communication with form of Ajax non-refreshing pages and the server-side still/also containing the business logic.)

RIAs operating in the context of the web-browser have thereby the limitations of the web-browser such as security limitations and policies, accessing system resources and the user interface from the web-browser as wrapper for the application.

A runtime environment independent from a browser is an alternative to avoid the limitations of the traditional web-applications. The access to the local resources is managed by the runtime environment. The security is the responsibility of the runtime.

An example of a RIA is the Gmail application from Google. Gmail is an online e-mail client that communicates asynchronously. It does not need to refresh the page to send or receive new e-mail. The search function makes it easy to find old e-mails and it is simple and easy to use as user-interface. Gmail cannot be used offline.

2.3 Investigating the potential RIA techniques

The next step is to introduce and describe the techniques and order them in categories. The goal is to categorize the techniques and to decide if they are capable of adding offline support to J-DigiNotes. These are chosen based on information and samples available on the websites of these techniques and trying out the capabilities of the techniques.

2.3.1 HTML-4 specification

HTML-4 is the most used publishing language for websites at the moment. The HTML-4 specification is recommended since December 1999. Traditional web-applications use the HTML-4 specification for the creation of web-pages. It has support for representation of text and media content as images and audio. It does not support offline storage. The HTTP protocol allows cookies to store a very little amount of data. It is not able to provide a rich interactive user interface. Many extensions for web-browsers are available to improve the user interface and allow multimedia content on web-pages.

2.3.2 HTML-5 Specification

HTML-5 is the successor of the HTML-4 specification. The HTML-5 specification is started by WHATWG (Web Hypertext Application Technology Working Group) in 2004. This specification defines the publishing language for the world wide web. The first public draft of the specification is released on 22 January 2008. It contains new features over the previous HTML version. One of the new futures is the local storage and the database storage. Web-browsers which are HTML-5 valid are able to store data locally. The top three most used web-browser vendors will release a version of their web-browser with the new HTML-5 feature enabled.

So far Firefox 2, Firefox 3, Internet Explorer 8 and Safari 3 have implemented a part of the specification and offline storage feature is already available. The HTML-5 specification does not tell the web-browser vendors how to implement the offline storage so each vendor can have different ways of storing data.

The offline store from one web-browser will not work for another web-browser. According to the timeline HTML-5 should be recommended by W3C at the end of 2010. The development is divided in five milestones and the first working draft milestone has already been delayed. The estimation from the WHATWG writer is that HTML-5 will reach W3C candidate recommendation somewhere during 2012. Until then, parts of the specification can be used but won't be widely supported. HTML-5 is a possibility to use as a way to migrate the web-application to add offline functionality.

2.3.3 Gears

Gears is a product from Google to extend the web-browser to let developers create web applications that can partially operate offline. It is currently an early-access developer's released product under the BSD license and can be used and modified without restrictions. The key features are described as: a local server, to cache and to serve application resources without needing to contact a server. The second feature is the local database, to store and access data from within the web-browser. Gears also provide a worker pool. Intensive tasks can be run in a separate thread in the background without disturbing the main thread. By not disturbing the main thread on the client makes the web-application more responsive and positively influence the user-friendliness.

Gears operates within the sandbox of the browser. The sandbox is a secure controlled environment where plug-ins can operate within the boundaries of the sandbox. Like the HTML-5 storage, the location where Gears will store data is different for each the web-browser.

2.3.4 Dojo Offline

Dojo Offline is an open-source toolkit released under BSD license written in JavaScript. It is a module of the whole Dojo Toolkit. The Dojo Toolkit provides client-side functionality for web applications. The offline module is built on top of other offline storage providers. It offers the same functionality on a higher abstraction layer. It will first check if the client browser has Google Gears installed, then it will use that as storage provider. The second choice is Adobe air-runtime as local storage provider and the third choice is DOM-storage from HTML-5 compatible web-browsers. Dojo offline is an abstraction layer and is dependent on at least one of the underlying techniques offering offline storage.

2.3.5 Prism

Prism is a beta product from Mozilla, formerly known as WebRunner. Prism is an experiment to bring 'web-applications closer to the desktop'. Prism is based on the concept of a site-specific browser. It uses the Gecko layout engine for rendering. This engine is used in the web-browser Firefox, and Thunderbird.

Basic desktop integration enables users to start the web application from a shortcut with an icon in the start menu or on the desktop. Prism does take a web application closer to the desktop with separated process and a startup icon, but prism does not provide any offline support so far.

2.3.6 AIR

AIR is the acronym for Adobe Integrated Runtime and was originally named Apollo. AIR is a multiplatform runtime environment that allows developers to deploy RIA's on the desktop. AIR is able to run applications based on HTML/JavaScript, Adobe Flex and Adobe Flash. Flash applications can also run in the web-browser, but AIR is not bound to the limitations of the browser because it operates in its own runtime environment. AIR supports offline storage and allows applications to work both offline and online. The AIR environment gives the opportunity to use the AIR-functionalities in the RIA via it's API. AIR can be used in the experiment, because the created RIA can use the AIR offline storage functionalities.

2.3.7 Silverlight

Silverlight is a technology from Microsoft to enable RIA's. Silverlight is a cross-browser and cross-platform plug-in for the web-browser. It is a competitor of Flash and JavaFX. Silverlight is build with a subset of the WPF (Windows Presentation Foundation) framework.

WPF is used to build rich user interfaces for desktop applications on top of the .NET framework of Microsoft. The strength of the Windows Presentation Foundation usage is that it allows 3D support in the API. In Flash, 3D support is something that has been lacking until version 10. For flash version 10 and later, additional 3D supported is added. Silverlight uses the new XAML language for the declaration of the presentation layer.

Silverlight 2 belongs to the RIA category, but is not able to store date offline at the client yet. A feature version of Silverlight, will include new features with local data storage.

2.3.8 JavaFX

JavaFX is an open source scripting language from Sun. It is a new language to create rich user interface applications. The JavaFX script operates in the runtime environment of java from version 5 and higher. It can run on the desktop and in the JRE sandbox in the browser like an applet. An advantage of JavaFX is that the Java runtime is widely used on personal computers and handheld mobile devices. JavaFX offers no offline functionality.

2.3.9 Overview of the techniques

The techniques have been introduced. Now the techniques are categorized to see which can store data offline and can be used in this research. Table 1 on the next page gives an overview of the techniques. The table consists of the following columns:

- Rich Internet Application; a RIA is defined as a web-application with an enriched more responsive user-interface. If the technique is able to create a RIA it belongs to the category. Air, JavaFX and Silverlight are the tree techniques that can be categorized as RIA-frameworks.

- Runtime on desktop; the technique operates in a runtime on the desktop. The technique does not operate within the web-browser. HTML-4, HTML-5, Gears and Silverlight all operate within the web-browser.

- Sandbox; the technique operates within the sandbox of the web-browser. The web-browser is the runtime and with an extension the technique adds functionality to the browser. The sandbox has managed security policies. HTML-4 and HTML-5 are specifications that are implemented by web-browsers render engine and are not related to a sandbox. JavaFX is based on the JVM. Gears, Silverlight and JVM are extensions to the web-browser. These extensions operate in the security sandbox of the web-browser with restricted access to system resources.

- Offline data manipulation; the technique is able to store data offline at the client and can use and modify the stored data. The data can be accessed without being connected to the server. HTML-5, Gears and Dojo Offline toolkit are able to open a page in the web-browser and manipulate local stored data. The AIR provides an API to manipulate local stored data for applications operating in the AIR runtime environment.

- Platform; the supporting environment of the technique. All operating systems means Windows XP, Linux, Mac OS X10.2 and higher.

- Data storage type; describes what types of storage is provided by the technique. The storage can be database-storage that can be accessed by SQL-language or file system storage where files can be accessed on the local hard drive or simple key-value storage.

Table 1: Overview of techniques

Technique	RIA	Runtime on desktop	Sandbox	Offline data-manipulation	Platform	Data Storage type
HTML-4	X	X	X	X	All	-
HTML- 5	X	X	X	Yes	All	Browser implementation of database and file system storage
Google Gears	X	X	Yes	Yes	All	SQLite (Database), LocalStore (File System)
Dojo Offline Toolkit	X	X	X	Yes	All	Gears / HTML-5 / AIR based storage
Mozilla Prism	X	Gecko engine	X	X	Windows XP, Linux	-
Adobe AIR	X	Adobe Integrated Runtime	X	Yes	Windows XP, Mac OS X 10.x	SQLite (database) File System
Microsoft Silverlight	Yes	X	Yes	X	Windows XP, Mac OS X 10.x	-
JavaFX	Yes	Java Runtime Environment	Yes	X	All	-

2.3.10 Offline Storage

The techniques are displayed in table 1. The techniques HTML-5, Gears, Dojo Offline Toolkit and AIR are able to perform offline data storage. These techniques exist for different goals.

The HTML-5 is a complete specification that the browser should implement in order to provide offline storage. The HTML-5 specification provides data storage as offline files and a SQL-database. These are accessible with by a JavaScript API. The web-browser is responsible for the implementation of the specification, and how and where the data is stored.

Gears is a web-browser extension with similar offline functionalities as HTML-5. Gears provides the offline functionalities as an extension to the most used web-browsers. The same offline functionalities as HTML-5 can be used in a HTML-4 web-application. Gears is responsible how and where the data is stored.

AIR is a separate runtime and thereby different from Gears and the HTML-5 specification. The runtime uses a security sandbox and allows offline data storage and local file storage. The offline storage functionalities are similar to HTML-5 and Gears. An SQL-Database and filesystem access is available.

The Dojo Offline toolkit supports offline storage is an abstraction layer on top of Gears, HTML-5 and the AIR implementations.

2.4 Result of investigation

RIA's can overcome the limitations of the traditional web-applications. We found out that not all techniques are suitable for the research. The techniques able to provide offline storage are Dojo Toolkit, HTML-5, Gears and AIR.

For the research, two techniques are selected to perform a migration of the JEE web-application. The Dojo Offline Toolkit is an abstraction layer for storing data offline within the web-browser; it is dependent on the Gears, AIR or HTML-5. Dojo Offline Toolkit *cannot* store data offline itself. It requires choosing one of the techniques which *can* store data offline. For the research the underlying techniques are selected.

HTML-5 will be recommended in 2010 or later and is not yet implemented by the most used web-browsers (Internet Explorer 7, Firefox 2 and Safari) at the moment. Some web-browsers have partially implemented the offline storage functionality. We have decided with Sogyo that the web-application should work on most used web-browser. That is why Gears is chosen over the HTML-5 specification.

Gears is an extension for the most common web-browsers. The extension provides a JavaScript API to create offline data and make use of a client side database. The distribution of the application is same way as the traditional web-application. The offline functionality could be added to original JSP's in the presentation-layer.

AIR is a runtime environment. AIR is able to run Flash and HTML/JavaScript applications. In the baseline the presentation-layer contains the HTML with layout inside the JSP. This HTML could be reused in the AIR application.

AIR applications have a different distribution than traditional web-applications. With AIR the applications becomes an installer-file 'J-DigiNotes.air' that should be installed at the client. The installation gives extra feature, a start-menu short-cut, that Gears does not provide. The extra features of both techniques will be discussed in the comparison chapter.

3 Experiment

The JEE web-application JDigiNotes is the baseline for the experiment where two of the techniques Gears and AIR, are applied to enable offline functionality. First we define what the offline functionalities are and when it is considered offline. After that is the architecture of the baseline explained. We describe how the migration is performed with both techniques. In the migration is explained what steps are taken to enable the offline functionality.

3.1 Scope of the offline functionalities

In this chapter is the scope described of the offline functionalities and what the term offline means. First to be clear what we mean by 'working offline':

- The web-application should be able to be started without being connected and be able to show the notes from the last synchronization.

If the client access requests a page and web-browser is not connected to the server than the page with the information cannot be shown. The web-browser will show that the page does not exists or the server cannot be reached. This is also happening for the baseline. Not being able to connect to the server can have different causes. The infrastructure between the client and server, the server is unavailable or the client has a problem. In the migrated offline case should the client be able to synchronize the notes if a connection is available.

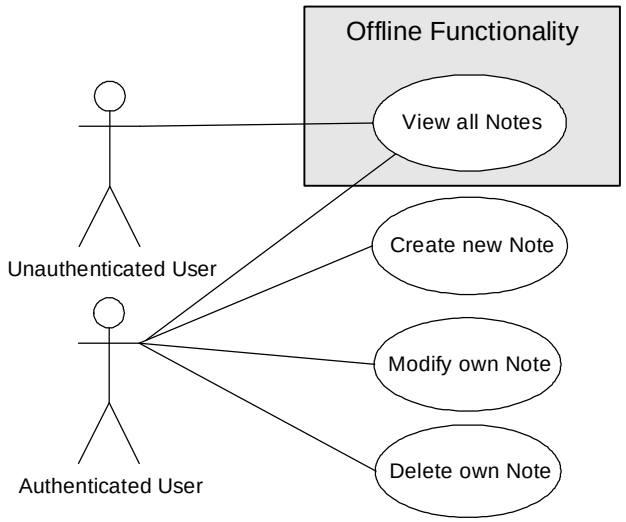

Figure 2: Scope of offline functionality added during migration

This figure shows what parts of offline functionalities are added during migration. The creation, modification and deletion of notes is not supported in the offline variant. A more complicated synchronization strategy has to be created to support those actions.

Another important point that should still apply after the migration is versioning. The web-application created with the techniques should be updatable to newer versions. A new version of the baseline does not require to the client to update the application manually. A new version creates new content or layout for the client. With a traditional web-application the client only have to refresh the current page. Changed files from the web-application with the techniques should be updated on the client without re-installing the application manually.

The notes on the server can be changed or added while a client is disconnected. The notes on the server and client will get different versions. To update the new and changed notes some kind of synchronization strategy is required. We can either choose Manual-synchronization or background-synchronization. The manual-synchronization strategy is a user or timed triggered event that starts the synchronization-process. A background-synchronization strategy is a process in the background of the application that continuously synchronizes the data between the local data store and the server without requiring user input. In the JDigiNotes web-application the manual-synchronization process is implemented, because its relatively easy to implement compared to background-synchronization.

The update-button should try synchronizing the client-side notes with the server-side notes of the web-application. If the web-application can be reached and the server is online, the changed and added notes should be added to the local database on the client. In a traditional web-application the user have to wait until the page is reloaded, before the new notes are visible and the user can perform a new action. The synchronization process on the web-applications should not block the user interface.

The architecture of the client side can be either modal or modeless. Modal applications have distinct offline versus online states, that is usually indicated through some change in the user interface. The user is made aware of the state and participates in switching states in some manner. Modeless applications attempt to transition seamlessly between online and offline states, without significant UI changes. [Gears architecture] The user does not need to participate in switching states, which the application does automatically. In the experiment the modeless architecture is chosen.

To enable the synchronization in the application the protocol AJAX is used for client-server communication. The AJAX implementation for the synchronization process influences both the client and the server. The client needs to be able to make a-synchronous calls to the server and the server must be able to handle the call with Servlets in the Controller-layer.

3.2 Architecture of the baseline

This section explains the architecture of the application. The web-application is based on the Java platform. The Java programming language is a requirement for the research. The Java Enterprise Edition 5 development kit is used to create the application. This edition is an extension of the Java Standard edition with additional features to create a web-application. It allows the use of [Servlets] and JSP's [Java Server Pages] to generate HTML web-pages for the client.

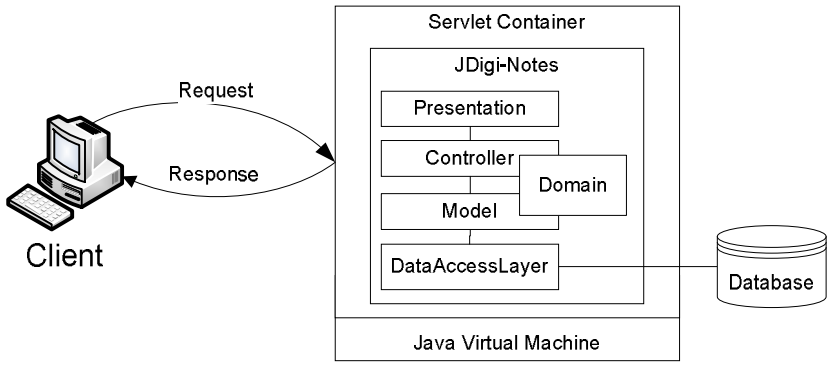

Figure 3: Architecture of J-DigiNotes

The web-application is running in a Servlet container. The Servlet container is able to run (multiple) Java based web-applications and can handle concurrent users. The Servlet container is also responsible for the sessions management of the users. The Servlet container uses the java compiler to compile the JSP's and Servlets to interpreted executable class files.

A high level of separation of concerns is encouraged by the MVC-pattern. By using this pattern the web-application is separated in different layers. Each layer is a component with its own responsibility. The separation of responsibility over the layers increases the reusability. The effect of a change in a part of the application should be minimized to a particular layer and not spread through the whole application. The reusability of the components is important for the research to test which technique can add offline functionality with high reuse. Each layer has a single purpose that supports high cohesion. The layers can only communicate with the connected layer. It is not the concern of presentation layer to store data so the presentation-layer cannot communicate with the data access layer, it can only inform the controller-layer.

3.2.1 Overview Layers

The MVC-pattern is used to divide the application in multiple layers to separate concerns and have high cohesion [Freeman 2004]. In Figure 4 is an overview of the layers with the corresponding implementations.

The view of the MVC-pattern is fulfilled by the presentation-layer. The purpose of the presentation-layer is to create the user interface and have no knowledge how to handle the actions from the clients. In the case the presentation-layer uses JSP's to generate HTML-pages and Cascading Style Sheets to create the user interface for the client. The actions performed on the HTML-pages by client are sent to the controller-layer as requests.

The controller-layer its purpose is to define the application behavior and select the view for the response. A controller is available for each function. The controller-layer is uses Java Servlets to handle the requests and responses from the clients and manages the http-sessions. Based on the request and the session from the client, information is retrieved from the model-layer and a response is created. The Servlet forwards the response to the corresponding JSP in the presentation-layer that will create the layout.

The model-layer exists of default java classes and is responsible for the business-logic and validation of the data. The model can access the data by communicating with the Data Access-layer.

The data access layer uses the JPA abstraction layer to have loose coupling between the model classes and Hibernate, the persistence unit. Hibernate is the JPA implementation and is used to communicate with the database.

The database can be many relational database variant because it is interchangeable. For the case HSQL-DB is chosen, because it supports in-memory database that improves testability. The in-memory database starts when the java tests are executed and is initialized with the test data without environmental influences like user-rights or changes in data between tests.

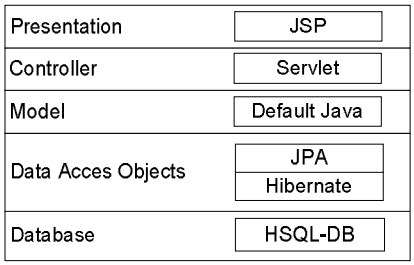

Figure 4: Layers and implementations as described above

This figure is an overview of the layers and there implementations used in the baseline. We gave an short introduction to the architecture of JDigiNotes. In the next paragraph we describe each layer top-down what its purpose is and how it operates.

3.2.2 Presentation-layer

The presentation-layer is responsible for generating the user-interface for the client. In this case the visual layout of the web-page is generated by the presentation-layer.

The visual layout of the HTML web-pages is created with the JSP's and Cascading Style Sheets. The JSP's are stateless and contain HTML-markup mixed with JSTL (java standard tag libraries) tags to support dynamic creation of the content. The JSP's *are able* to execute Scriplets, blocks of Java code between the markup, but that is *not* done in the case to have a clear separation between the layout and the handling of user actions and business logic.

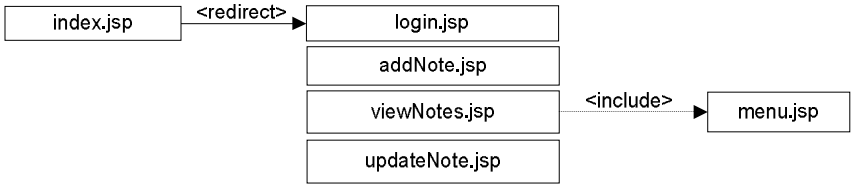

Figure 5: Overview of the JSP's

This figure shows the pages from J-DigiNotes. If the client enters the ULR of the web-application the ServletContainer opens the default page index.jsp. This page forwards the client to the login.jsp. The login page allows the client to enter the username and password. If the credentials are incorrect an error is shown on the page.

The page addNote.jsp is a page with an input field for the note content.

The viewNotes.jsp displays all notes in a table with the information: id, content, creator, creation date and update date. For each note a column delete and update exists. These columns have a clickable link to delete or update the note.

The menu.jsp contains links to add a note and logout from the application. The menu is included in viewNotes.jsp at the top of the page.

3.2.3 Controller-layer

The controller-layer is for handling of actions from the client. The actions are in form of HTTP-requests. The requests are sent to the corresponding Servlet based on the URL. The mapping between URL and Servlet is configured in the deployment descriptor. The deployment descriptor is a web.xml file containing the main configuration for the ServletContainer. In this case the configuration file contains the available Servlets. A combination of the servlet-name and the servlet-class define a Servlet. The Servlet is mapped to an URL-pattern. The configuration file does contain an exception that is mapped to an error page if the user is not authorized for the action. Figure 12 in the appendix contains a part of the web.xml.

For each action a user can do on the baseline *(login, logout, add, remove, update, view)* a different Servlet is available. The reason for this is to separate the responsibilities and have loose coupling between the classes.

The Servlets receive the request based on the URL and will generate a response for the client. For each client a temporary session is created on the server to store the person's identifier when the client logs in. The session on the server is stateful are keep track of the person's identifier for a number of minutes configured in the web.xml until the session is removed.

Restricted actions are first checked by the AuthenticatorServlet whether the request is from a client with an active session containing an existing person's identifier. If the person's identifier exists the Servlet may continue the action. The restricted actions are: add, update, remove a note. The unrestricted actions login, logout and viewing notes are not checked by the AuthenticatorServlet and do not require the client to have a session.

If the action on the Servlet is allowed to continue, it will gather the required information from the model-layer and create a response object. The response is sent to a JSP for the creation of the view. The forwarding based on the result of Servlet to a JSP's is displayed in Figure 6:

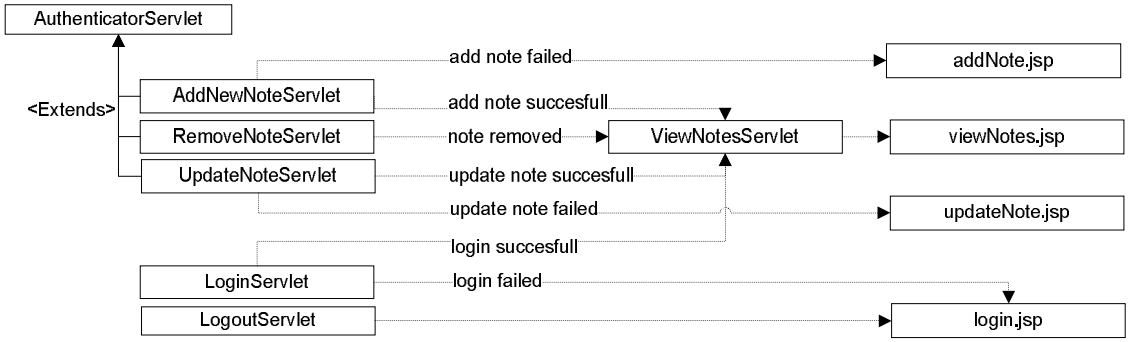

Figure 6: Servlets and relation to JSP's of JDigiNotes

For the AddNewNoteServlet, ViewNotesServlet, UpdateNoteServlet and the LoginServlet a corresponding JSP is available. If the AddNoteServlet failed to add the note, the client is forwarded to the addNote.jsp with an error message why the note was not added. If the note was added successfully the client is forwarded to the ViewNotesServlet. The ViewNotesServlet gathers all notes and uses the viewNotes.jsp to render the view for the client. The UpdateNoteServlet and LoginServlet work the same way.

The RemoveNoteServlet has the same viewNotes.jsp for a success and failed remove action. The failed action displays an error message why the note could not be removed.

The LogoutServlet destroys the session on the server and redirects the client to the login.jsp.

The Servlets extending from AuthenticatorServlet are only accessible by authenticated and authorized users. If the client has a session with a person's identifier the Servlet continues, otherwise an UserNotLoggedOnException is thrown and the client is forwarded to the login.jsp page. The exception and forwarding to the login.jsp page are configured in the web.xml.

3.2.4 Model-layer

In the baseline application J-DigiNotes the Servlets have the responsibility to handle the requests and control the user sessions. The Servlets use the model-layer to retrieve the information. The models-classes are stateless, independent from the controller-layer and know how to retrieve the information from the data access-layer. There are two models the NotesModel and the PersonModel.

The NotesModel is used by the Controller-layer to add new or updated notes and stores and retrieves notes. The NotesModel validates the input of added and changed notes before it is persisted. The PersonModel is responsible for checking credentials of the person. The result of an action from the NotesModel or PersonModel is sent back to the Servlet in the Controller-layer.

Person management tasks (add, update, delete persons) are not part of the case. These tasks are executed directly into the database with a database tool.

An example of the login procedure: The user filled in the username and password on the login.jsp. The request is sent to the LoginServlet. The LoginServlet asks the PersonModel if the entered credentials are valid. The PersonModel retrieves the person from the data access layer and check the credentials. Than it sends a message back to the LoginServlet that the credentials are valid or invalid. Depending on the answer the LoginServlet creates a session and the client is redirected to the ViewNotesServlet or back to the login.jsp with an exception. This redirection of the Servlet to the JSP is visualized in Figure 6.

3.2.5 Data Access Layer

The data access-layer is responsible for retrieving and persisting of the data. A database is used to store the data. A database can be queried to retrieve information from it. The data is stored in the relational SQL-database [HSQL-DB]. It is an open source database engine that supports the SQL-1992 standard and can be used as an in-memory database for testing purposes.

The web-application uses the JPA (Java Persistence API) as an abstraction layer between data access layer and database. This makes HSQL-DB interchangeable with other relational database variants. JPA is part of the Java 5 EE specification.

The relation between the database and the objects in the Java is done by the object-relational-mapping (ORM) framework [Hibernate]. The framework provides the coupling between the entities and database scheme and allows SQL queries to retrieve collections of entities. The framework requires mappings to have knowledge about how to convert the objects to records in the database. The mappings can be configured in XML-format as separate files or as annotations in the entity classes. The downside of XML instead of annotations is that they introduce extra files in the source code. The downside of annotations is that they are inserted in the entity classes and can have a negative effect on the readability of these classes. The functionality and possibilities of both options are the same. In this case the location of the mapping is done by annotations in the entity classes following the JPA guidelines.

3.2.6 Relation to PetStore

The PetStore is an open-source reference from the BluePrint-program from Sun to show how JEE can be used to develop web-applications. This reference is used to validate the JDigiNotes application. To measure how representative the JDigiNotes application is.

The PetStore has different versions and flavors. In this research the official PetStore based on JEE 5 from the Sun is used [Java Pet Store 2.0]. This matches the Java version used for JDigiNotes. The PetStore has more features than JDigiNotes but has common architecture.

The layers as we have described above can partly be found in the PetStore. The packages in the PetStore are not named as these layers but organized differently. The presentation-layer is implemented with JSP's. For each action a different JSP is available like JDigiNotes. The PetStore uses code-blocks instead of JSTL-tags that are used by JDigiNotes to generate the dynamic content in the JSP's.

JDigiNotes has controller-layer that contains all controlling Servlets. PetStore uses one Servlet to handle most tasks and JDigiNotes used multiple Servlets with each a different task. In the PetStore the servlet is placed in the controller package with other classes that offer validation, autocomplete and fileupload utilities. The PetStore sends each action to a JSP. The mapping between the URL and corresponding JSP is done in the Servlet itself in a key-value map. JDigiNotes uses the ServletMapping property in the application descriptor (web.xml). Both web-applications use the Servlets handle the request and sends the response to the corresponding JSP based on the URL.

The PetStore uses a façade pattern [Freeman 2004] in the Servlet to retrieve the data. The façade represents the data-access-layer in the JDigiNotes application. Both use the Java Persistence API to store and retrieve the objects from the configured data-source. The data-source configuration is located in the persistence.xml. The JDigiNotes application has a model-layer between the controllers and the data access-layer. The controller-layer mainly handles the session from the client and checks the login session state. The PetStore does not use login functionality and is thereby not required to validate the login session state.

The entities are located in the model package in PetStore. The JPA-mapping is done with annotations on the entity classes instead of mapping the entities in xml. This is the same approach used in JDigiNotes. A noticeable difference is that the validation of inserted data is done inside the domain objects of the PetStore application. In the JDigiNotes application this logic is done inside the controller of the performed user-action.

The presentation-layer of PetStore uses the same view-technique as JDigiNotes. It is implemented with JSP-pages and JavaScript in the web-package.

JDigiNotes is a typical JEE web-application as they are created within Sogyo. It uses the same guidelines and patterns. It has more or less the same architecture as the PetStore with less functionality and Ajax support. The PetStore is a often used as a reference for comparing and validating.

3.3 Migration with Gears

Gears is the first technique used in the migration. The requirements for the offline functionalities of the web-application are able to view notes offline. Synchronize the new notes if a connection is available. And being able to persist new versions of the web-application to the clients. These need to be implemented on the baseline with Gears. This chapter describes how the migration is performed and which steps have been taken.

Gears operates in the web-browser and can use JavaScript frameworks to support the client-side functionalities that are supported by the web-browser. It uses the Gears extension in Firefox and Internet Explorer. The gears_init.js is provided by Google to obtain access to the Gears factory and APIs. An example how Gears is included in a page is shown in the figure below:

```
<html>
<head><title>Gears Example</title></head>
<body>
<script type='text/javascript'
        src='../gears/gears_init.js'/>

<div id='out'>[overwriten]</div>

<script type='text/javascript'>
var outputDiv = document.getElementById('out');
if (!window.google || !google.gears) {
    outputDiv.innerHTML = 'No Google Gears';
} else {
    outputDiv.innerHTML = 'Google Gears installed';
}
</script>
</body>
</html>
```

Figure 7: Example: enable Gears in HTML / JSP

The web-browser displaying the page will load the Gears JavaScript file and execute the JavaScript block. In the example Figure 7 will the script code-block overwrite the text of the div-element with new text, based on the availability of Gears.

The resources required by the ViewNotes need to be available offline. Gears provide a ManagedStore to store the required files on the clients local machine. The ManagedStore can be created from the LocalServer from the Gears API. The ManagedStore is able to store and delete files on the client machine. The resources required for viewing the ViewNotes page are:

- The HTML page, containing the layout.
- The layout.css for the visual style of the HTML page.
- The image that is used on the page.
- The gears-init.js to enable the Gears-API.
- The update-notes.js that implements use offline functionality.
- The used Ajax framework files.

The ManagedStore is configured to read a manifest that tells what files are used for the offline functionality. The figure below shows how the manifest is implemented in the JDigiNotes application so Gears can use it to manage the entries:

```
{
  "betaManifestVersion": 1.0,
  "version": "firstRelease",
  "entries": [
      { "url": "viewNotes.html"},
      { "url": "style/layout.css"},
      { "url": "pitures/remove.png"},
      { "url": "javascript/update_notes.js"},
      { "url": "javascript/gears/gears_init.js"},
      { "url": "javascript/dwr/interface/NotesController.js"},
      { "url": "javascript/dwr/engine.js"},
    ]
}
```

Figure 8: manifest.json containing all resources for the ViewNotes page

This file is formatted as JSON (JavaScript Object Notation). JSON is a lightweight data-interchange format. It became a standard part of ECMA JavaScript in 1999. It is easy to read and write by humans and easy to parse by computers.

The files are listed in the collection entries in 'manifest.json' combined with a version. Gears uses the file to check the current version. If the version of the manifest is increased, all entries are downloaded again from the server. The 'manifest.json' is available in the web-directory on the server and is requested each time the client opens the viewNotes page. If the gears application is updated the manifest is changed and the version attribute is increased. All client will download the new sources. With this mechanism updates of the web-application can be distributed to the clients.

Making the view-notes functionality work on the client requires these steps:

- Generate the content in the view, update the HTML-page on the client-site.
- Persist and retrieve notes and persons on the client-site with Gears.
- Synchronize the notes with the server by Ajax call.

The first step is to generation the content in the view. ViewNotes.jsp contains the dynamic creation of content for the notes. The client should be able to generate the content based on the stored notes. The JSP-pages contain JSTL-tags that are used to generate the dynamic content inside the HTML-layout. The notes are added to the page and the page is sent to the client. JSP and JSTL-tags are used in the ServletContainer and cannot be used on the client. The creation of the content for the ViewNotes pages is replaced from JSTL tags to JavaScript. The JavaScript code can be executed at the client in the web-browser. After the migration the representation of the notes is still in HTML-tags.

The second step is persisting and retrieving the notes on the local machine. To be able to display the notes offline, the information of notes and persons should be available on the client.

The domain objects Person and Note are retrieved on the client in JavaScript Objects. Figure 9 shows a collection of two notes in JavaScript objects assigned to a variable *notes*. The object available at the client

should not contain all the properties from the original domain that is on the server-side. A person contains a password that should not be available at the client. Otherwise the passwords could be accessed by other users of the application.

The Ajax framework for this project is Direct Web Remoting [DWR]. DWR is created to have an easy bridge between JavaScript calls and Java. The advantages of the framework are the easy configuration and controlled exposure of classes and methods. The conversion from domains objects to JavaScript objects is realized by DWR. DWR is configured in dwr.xml.

To make the DWR available in the project the JAR is added to the classpath and a servletMapping is configured in the web.xml so that the path /dwr/* is redirected to the DwrServlet. The DwrServlet uses the dwr.xml configuration to map the java objects to JavaScript object and expose methods.

To encourage object orientation the notes and persons are treated as domain objects just like the model-layer on the server-side. The notion of a Note and a Person object gives the ability to extend the offline functionality and support maintenance.

```
var notes = [
{   id: 1,
    content: "note content",
    creation: "12-03-08 12:41:40",
    creator: {  id: 1,
                name: "Alexander"},
    updated: null
},
{   id: 2,
    content: "the last node",
    creation: "14-03-08 16:30:21",
    creator: {  id: 1,
                name: "Alexander"},
    updated: "14-03-08 16:45:30"
}];
```

Figure 9: JavaScript object-array of two notes with a creator

Notes should be updatable with an update action. The action is triggered by a user that clicks the update button. If there is no connection to server, the update call will never be answered and the notes will not be updated.

3.4 Migration with AIR

This section describes how the baseline is migrated to the offline variant with AIR as used technique. The migration with AIR is different from the migration with Gears. The AIR runtime environment can only execute a compiled *.AIR file as web-application and Gears applications stay in the web-browser.

An AIR web-application can be created with HTML and JavaScript. In the baseline the presentation-layer contains the HTML-markup inside the JSP's. The markup can be reused from the baseline in the AIR web-application.

The application created in HTML and JavaScript is packed into an AIR file with a commandline operation. The WebKit engine used in Adobe AIR allows most Ajax frameworks to work in Adobe AIR. Adobe AIR has a multi-tier security model that only allows access to desktop APIs to code that is run within what is known as the "application sandbox". To enhance the security of AIR applications built with HTML and JavaScript, Adobe AIR has restrictions on the application sandbox for JavaScript applications. These restrictions prevent some Ajax frameworks from working in the application sandbox. The site contains a table explaining which framework is supported and what the restrictions are. [supported AJAX frameworks]

The DWR framework used in the Gears migration cannot be used in the AIR migration. The framework MooTools is chosen because it has no conflicts with the AIR security sandbox. The MooTools JavaScript framework enables AJAX requests with JSON data format.

The ViewNotes.jsp containing layout is subtracted and used to create a new HTML file. The JSP and JSTL tags are removed, because the file is going to run it the AIR runtime environment and does not support these tags. The dynamic creation of the HTML page with the notes is handled with JavaScript. The JavaScript has to be added to the application.

Notes synchronizing

The client application is based on JavaScript that works well with the JSON format. The client generates a JSON call with MooTools. The call is sent to the server requesting the new notes since the last updated note. If the call reaches the server, it will collected the found entities. Otherwise the update procedure will end.

On the server an new Servlet is introduced that deals with the JSON-requests from the client. It parses the request and after gathering all data it will replay to the client request. A method on each domain object is introduced that converts the current object into a JSON-object. The collected domain objects Persons and Notes are converted to JSON with such a method and sent back to the client. What attributes are available in the JSON-object is decided in the conversion method. The Persons password will not be converted.

4 Results

This chapter contains the results from the migrations with the two techniques. The results of the migrated versions are measured in LOC (Lines of Code) and in changes of the architecture. The comparison of the results is discussed in the next chapter.

LOC is defined as a line of code in a file excluding comments and blank lines. In case of HTML and JSP start and end tags are included. A small Ruby script is created to count LOC of files for a given folder recursively. The script uses the file extension to determine the appropriate comment type. Each file matching a familiar file type is read and valid lines are counted. The familiar file types are Java, JSP, HTML, XML and JavaScript. The scripts is executed for each project (baseline, gears and AIR) on the root of the project.

4.1 Baseline Measurement

The measurement of how much could be reused when migrating with a certain technique is performed with the amount of source code added and changed from the source code of the baseline. The source-code of the baseline is measured. The LOC are divided over the different layers and packages:

Table 2: Layers with corresponding LOC of the baseline

Layer	Package	Java	HTML JSP	XML	Java Script	Files
DataAccessObjects	nl.sogyo.notes.dao	20				2
	nl.sogyo.notes.dao.jpa	149				2
Model	nl.sogyo.notes.model	121				2
	nl.sogyo.notes.model.domain	92				3
Controller	nl.sogyo.notes.web	185				8
Presentation	webapp.pages		109			6
	webapp.script				0	0
(Configuration)	resources.META-INF			17		1
	webapp.WEB-INF			67		1
Total		567	109	84	0	25

Data Access Layer:
In the table we see that this layer contains 4 files. The *dao*-package contains two interfaces as the contract for the model-layer. The *jpa*-package contain the two implementations of the interfaces with the JEE standard JPA to access the database. The interaction with the database consumes the most lines of code here.

Model-layer: This layer containing the domain objects the most lines of code, because it contains the validation classes and the mapping to the database is included in the domain entities with annotations.

Controller-layer: The controller-layer contains the most files. The Servlets are described in the chapter 3.2.3 Controller-layer. In the table we see that the web-package have 185 loc. The created Servlets have logic of handling requests and forwarding to the correct page that is the biggest part of the code.

Presentation-layer: The presentation-layer exists of JSP-pages only. The JSP page mainly exists of HTML with extra JSP tags for the dynamic content. No JavaScript is used in the baseline.

Configuration: These 84 LOC came from the configuration of the web.xml Deployment Descriptor for the ServletContainer and the persistence.properties for the JPA configuration.
We have chosen an application with a small domain and that correlates to the size of the codebase. This can be helpful for measuring the differences with the migrated versions.

4.2 Gears

The first technique applied is Gears. Using Gears requires a web-browser extension that is available for the major web-browsers. To use Gears the JSP's have been modified with additional JavaScript code to replace server-side functionalities of the JSTL-tags to client-side functionalities.

The framework DWR has been used to enable AJAX calls from the client. The DWR-framework makes Java methods on the server-side accessible to the client in JavaScript. This requires an additional configuration file on the server. The configuration contains the access rights of the clients to classes and methods.

The files and LOC counted in the figure below are files created for the application by the developer excluding the files of the used frameworks. The empty fields mean there are no LOC for that part.

Layer	Package	Java	HTML JSP	XML	Java Script	add lines	change lines	delete lines	new files
DataAccessObjects	nl.sogyo.notes.dao	22				2			
	nl.sogyo.notes.dao.jpa	161				15			
Model	nl.sogyo.notes.model	126				5	4		
	nl.sogyo.notes.model.domain	92							
Controller	nl.sogyo.notes.web	185							
Presentation	webapp.pages		129			5	25		
	webapp.script				224	224			2
(Configuration)	resources.META-INF			17					
	webapp.WEB-INF			107		15			2
Total		586	129	124	224	266	29	0	4

Table 3: Layers with corresponding LOC of migration with Gears

To make the update functionality work a new method is introduced to retrieve the modified notes since a given timestamp. The method in the Model-layer influences the dao-layer that is should be able to support this call. That's why the model-layer and dao-layer have modified files and additional LOC.

The controller-layer is not modified. The use of DWR-framework simplifies the handling of remote AJAX-calls to Java methods. The call from the client in JavaScript is redirected by DWR to the method on the server. The result of the method is converted to a JavaScript-Object and returned to the function in JavaScript. This part is accomplished by DWR that does not require additional source-code.

Four new files created for the web-application. The update_notes.js file is the JavaScript file added to the Presentation-layer. Gears_init.js is added, but the LOC of gears s not counted because it was part of the used Gears- framework. The file is created to support the client-side functionalities and gears-storage. It uses the Gears-API and DWR-framework.

In the WEB-INF directory two files are added. The dwr.xml is added for the configuration for the DWR framework. The manifest.json is added to support new versions of the client application.

No lines of code have been deleted by applying Gears. This is because Gears is an addition to the existing web-application. It uses the existing controller, model and data-access-objects. The presentation layer is changed the most with reuse of the existing pages.

Change in architecture:

The introduction of DWR introduces an extra Servlet-mapping*(not a Servlet)* on the server pointing to the DWR-Servlet. The DWR-Servlet handles the AJAX requests. The way of communicating between server and client is changed. The client can now perform AJAX-requests for updating notes and the old requests for posting new notes. The Servlet-mapping is added to the deployment-descriptor (web.xml) and does not require additional source-code.

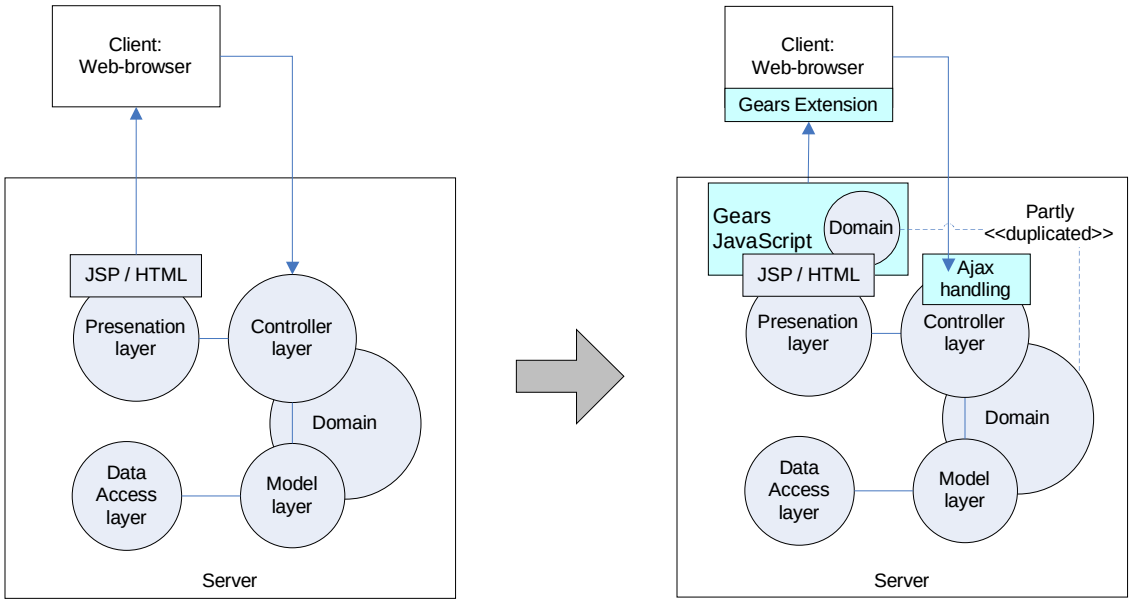

Figure 10: Baseline migration with Gears

The figure shows the software components of the baseline before and after the migration. The solid lines are interactions between the components. The dotted line is information related to those components. The round components are the layers of the application. The interaction between the client and server is to different layers. The layers and interactions have been described in chapter Architecture of the baseline.

The left side of the figure is the baseline and right is the migrated version with Gears. The right side of the figure shows the additional software components in migrated version.

All layer still exists after the migration and could be reused for the most past. As is visualized on the right side of the figure we see that the presentation-layer and controller have been extended. The extension adds logic to that particular layer. The Ajax handling introduces a new Servlet. With adding Ajax to the application we want minimize the transferred data from client to server. We want only to send the new notes that the client does not have yet. In the baseline all notes were required to build the page. The Ajax call for updating notes is based on the last updated note-date. To support this functionality it is added to the data-access-layer. That is why no lines have been added and not deleted at this layer.

The presentation-layer is changed the most, because the dynamic created content differently. The JSTL-tags have been replaced by JavaScript. The extension of the web-browser allows the additional Gears code to store data and files offline. The offline functionality to do this is added to this layer.

4.3 AIR

The LOC counted in the figure below are files created for the application by the developer excluding the files from the AIR-framework. The counting is done in the same way for the baseline and Gears. The ruby-script is executed on the top directory of the project and the following results appeared:

Layer	Package	Java	HTML JSP	XML	Java Script	add lines	change lines	delete lines	new files
DataAccessObjects	nl.sogyo.notes.dao	22				2			
	nl.sogyo.notes.dao.jpa	161				15			
Model	nl.sogyo.notes.model	126				5	4		
	nl.sogyo.notes.model.domain	111				20		1	
Controller	nl.sogyo.notes.web	237				55	1	3	
Presentation	webapp.pages		129			5	25	44	
	air.html		39			39			1
	air.scripts				277	277			3
(Configuration)	resources.META-INF			17		0			
	air			27		27			1
	webapp.WEB-INF			75		8			
Total		657	168	119	277	403	30	48	5

Table 4: Layers with corresponding LOC of AIR migration

As the same with Gears, AIR uses the 'update since a given timestamp' method the retrieve the latest modified notes. This method is used by the Ajax remote call. The implementation of the method is same as the Gears implementation. It influences the DataAccessObject layer and the controller. There is no difference between the Gears and AIR migration at this point.

A minor change is in the model-layer where the conversion of domain objects are added. This was required to have the Java objects converted to JSON.

The major changes in the code are at the presentation-layer. The JSP's are replaced by the HTML files and the dynamic content is generated by JavaScript instead of JSTL-tags in the JSP's. The HTML and JavaScript files are used by the AIR-compiler to generate the JDigiNotes.air file. The JDigiNotes.air is the application that will be running on the client.

To make this happen extra configuration files are added in the air package.

Change in architecture:

The change in architecture describes how the migration changes the original architecture.

The client-server pattern still applies in the architecture. The web-application operates in the AIR runtime environment and not in the web-browser. The change to AIR as the client involves a change the application distribution.

The files required offline are packed in the JDigiNotes.air and are installed on the client. Updates of the application can be distributed with the new air file. The air file contains a application descriptor with a version tag. The AIR runtime environment checks the version of the current running web-application before startup. If the version is outdated the user is notified and an updates process of the client application starts automatically.

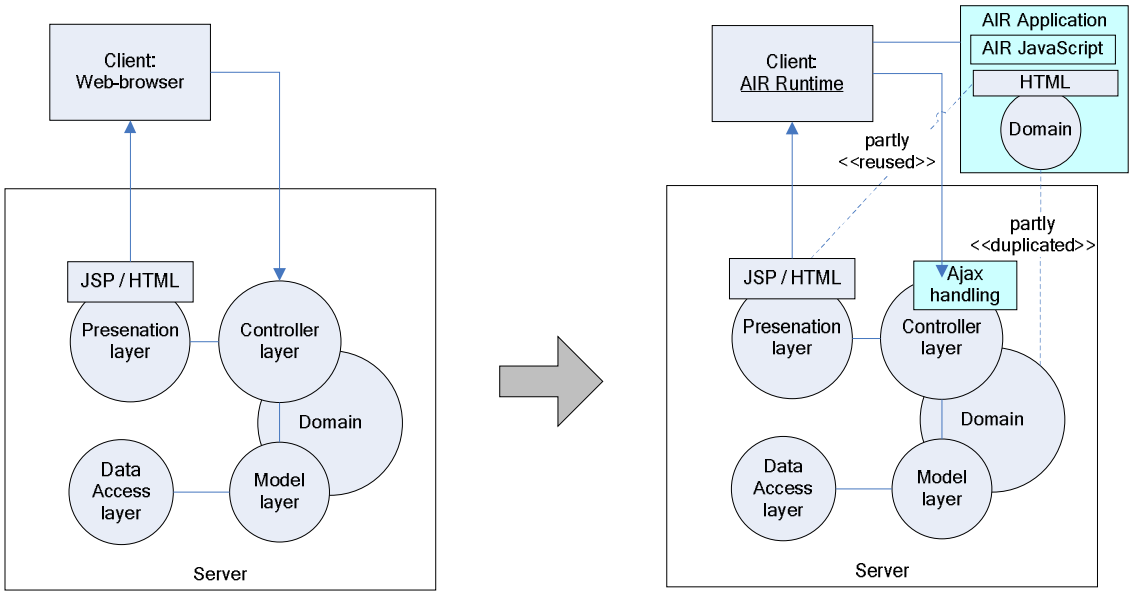

Figure 11: Baseline migration with AIR

The figure shows the software components during the migration from the baseline on the left to the web-application with AIR on the right. The client is changed after the migration from the browser to the AIR runtime environment. The runtime uses the created AIR application. The AIR runtime environment can run Flex, Flash and HTML with JavaScript applications, which is happening in our case.

As with Gears the layers are still present after the migration. The big difference is the presentation-layer. The JSP / HTML view is shifted to an AIR application file. The AIR application includes HTML-pages, JavaScript files and the HTML-resources: images and the cascading style sheet. The HTML files and resources can be reused from the JSP-pages from the baseline application. The presentation-layer is extended with the AIR offline support. The domain is partly duplicated in the JavaScript application to be able to store and retrieve the notes and persons.

The communication between client and server for updating notes is changed to Ajax-calls to support an uninterrupted user interface while synchronizing the notes. This change influences the controller-layer and data-access-layer. This is no different from what we have seen with the Gears migration.

5 Comparison

In the comparison the results are compared to each other. The differences are discussed and explained.

If we compare Gears to the AIR migration we have to point out the difference in type of client. In the migration with AIR the client is changed from the web-browser to the AIR-runtime environment. The AIR environment has a security sandbox to deal with which leads to limitations that are not encountered with the baseline and gears. An air-install file is downloaded and executed before the web-application starts. The use of the AIR environment influences the distribution of the application. The client application is downloaded as a separate file and installed through the AIR environment before it can be used. The air-install file contains the HTML files which content is partly reused from the JSP files of the baseline.

The migrations use AJAX to update the notes asynchronously. Each migration uses AJAX in a different way. The Gears migration uses the DWR-framework to support AJAX. DWR is a client and server solution for AJAX in Java web-applications. This adds configuration to enable the conversion of objects and access to methods. The AJAX in the AIR-migration is implemented with MooTools-framework at the client. At the server AJAX is implement manually in a new Servlet. No extra configuration files are introduced. It did introduced additional source code in the domain-layer for converting domain object to JSON-objects. The JSON objects are used on the client and are supported by MooTools.

The DWR framework is a convenient solution for the AJAX-calls that does not introduce an extra Servlet and does not require a manually created converters, because these are generated by the framework. The initial plan was to use DWR for AIR as well for a better comparison, but the AIR security sandbox does not allow some functions used in DWR. This is why MooTools is used as an alternative.

The client update-notes functionality in the JavaScript is different for both migrations. In the appendix in Figure 13 and Figure 14 the Gears and AIR variants of the note update sequence is shown. If we compare the files we see that in both solutions requestNewNotes() function is triggered by a button. The lastUpdated note date is retrieved from the local datastore. Both api's have a different way of accessing the local database:

The Gears-API provides a database object that should to be initialized before use. Queries can be executed against the database object and return a ResultSet object. The ResultSet contains rows of data like a ResultSet used in Java. On each row field() can called to retrieve its content. This method is very straight forward.

The AIR variant has a different approach. First a connection is created to the databasefile. After the connection is opened it can be attached to a statement. The statement is provided with a query with parameters and must be provided with the database connection. Than the statement can be executed and the result object can be requested from the statement object. The result object uses keys defined in the SQL-query.
"Select NOTE_ID from note" will give the resultObject from the statement. For example the id of the first result can be retrieved with resultObject.data[0].NOTE_ID, where NOTE_ID is the identifier.

The update functionality is started in requestNewNotes() that uses the lastUpdatedNote from the localDatabase. This value is used as a parameter to send an AJAX request to the server. Gears uses the DWR framework to generate the AJAX-request. The NotesController object is initiated by DWR and the exposed method are available this file contains the server url. The AIR-migration uses the MooTools to create a Request.JSON object. In the Request.JSON object the URL of the request is specified. Both variant have a callback function 'updateNotesCallBack()' incase a response arrives from the server. The different AJAX strategies has no influence at the client. It does not use the AIR or Gears API.

The updateNotesCallBack differs in format of returned objects. The DWR (from Gears) retrieves JavaScript-objects to the client as displayed in Figure 9. The MooTools request returns JSON-objects that are returned to the client.

Table 5: Overview of the differences of the migrations

Total	Baseline
Java	567
HTML / JSP	109
XML	83
JavaScript	0
Total	**759**

Gears	Reuse	Growth
586	99%	3%
129	77%	18%
123	100%	48%
224	-	-
1062	96%	40%

AIR	Reuse	Growth
657	98%	16%
168	33%	54%
119	100%	43%
277	-	-
1221	89%	61%

The table shows the LOC summary, reuse and growth of the package for each web-application. The reuse is based on the baseline minus the changed and deleted lines. This is divided by the baseline to have the result in percentages. The added lines are not included in the reuse calculation, but these are counted in the growth.

What do these numbers tell us?

It is a small project with a clear scope. The application is developed according to the standards of the company and compares to the guidelines of the PetStore blueprint from Sun.

The migrations with AIR and Gears have a great different reuse percentage on the presentation-layer. This is because in the AIR migration a part of the HTML/JSP is shifted to an AIR application file. The AIR file is based on the HTML and JavaScript. A part of the JSP could not be used. Only 44 lines were deleted to create the decrease to 33%. The Gears migration only JavaScript is added to the original code and the JSTL-tags lines were changed. With the migration to an AIR file the JSP pages could partly be reused, not as much as the Gears migration.

The AIR migration has grown more than the Gears migration. This is for the most part caused by the different used supporting frameworks.

Common

The domain objects (Note and Person) are required on the client. They are stored and retrieved locally. They are duplicated from the server domain objects into the JavaScript client-side functionality. The conversion of the domain objects in Gears are generated by DWR and in AIR is accomplished by a custom implemented conversion strategy.

The client side functionalities are implemented in JavaScript.

6 Evaluation

In this experiment we tried to point out the technique that can be used best to add offline functionality to a JEE Web-application in such a way that most code of the baseline application could be reused. To have a representative baseline we created the web-application JDigiNotes is based on the way Sogyo created JEE applications. And JDigiNotes is compared to the PetStore , a JEE reference implementation developed by Sun.

Besides the creation of the web-application techniques supporting the migrations where selected. During the investigation the techniques Gears and AIR came out to be able to support the migration. After applying these to the baseline we measured the results. As we have seen in the results the numbers do not differ that much. By applying a feature-change to the tree variants of the application we tested the impact of change.

6.1 Reuse

As we have seen in the results a few lines where removed, some are changed and the most were added. This happened for both migrations. It does not make a lot of difference which technique is chosen. We evaluate the results to point out what can be done differently to have better results.

By separating the application in layers the impact of the migration was for biggest part related to the presentation-layer. Both AIR and Gears can only be applied to the client-side of the application as we have found out. The other layers are there to support the change in the presentation-layer. In the supporting layers we added support for Ajax on the server-side for both migrations. This is done in a different way for each migration. This is the main reason why the migrations differ for the most part. Implementing Ajax in Gears is done with DWR, this is a simple effective way to support Ajax that was not possible in AIR. The limitations of the security sandbox of AIR does not allow all JavaScript-frameworks to work. We found this out after the migration was performed in Gears. AIR uses MooTools one of the supported frameworks. The advantages of DWR does not apply here and that leaves us with custom conversion of the domain objects.

For more clear results of the differences for both techniques the same supporting technologies and frameworks should be used. We have created a test-page for both techniques to get familiar with it, but did not apply Ajax to it. This is a crucial part of the migration. The test-page should include the crucial parts of the migration so there should be no unforeseen obstacles.

Besides the supporting techniques and frameworks we noticed that both techniques are applied to the frontend or client-side of the application. A web-application with a larger presentation-layer should increase the use of Gears and AIR and help point out the differences.

7 Threats to validity

The research is created based on the idea to make functionalities of a JEE web-application work on the client in an offline state. First technologies have been researched whether they are able to support the migration. Gears and AIR are used to support the migration.

Sogyo did not have a JEE web-application available for the research. The J-DigiNotes web-application is created as the baseline for the migration. The baseline architecture is created based on standards used in JEE applications within Sogyo and in the PetStore sample from Sun. The J-Diginotes web-application consists of a small domain that contains multiple entities with relations to each other. Large domains have many entities and relations, but the concept is the same. The PetStore contains multiple solutions to for the functionality. The solution using the MVC-pattern also commonly used in projects developed at Sogyo is compared to the J-DigiNotes. The solutions with other techniques are not used in the research.

The supporting techniques AIR and Gears are new in use. The development speed and quality of the solution is influenced by the understanding of the techniques. The better the technique and its features are understood the more elegant solution can be applied to a problem. To reduce the learning curve of both techniques, each technique has been used to create a proof of concept before applying the technique to the baseline.

AIR can use different solutions. For this research we have decided HTML and JavaScript as client-side application, because it is close to the original application. HTML can be reused from the JSP files and JavaScript could be added. The HTML and JavaScript migration of AIR is comparable to the Gears migration with JavaScript. The other solutions (Flash, Flex, ActionScript) instead of HTML and JavaScript should not influence server-side code, because it only applies for the client.

The measurements about the code are correct, because they are automated with a script and tools. The LOC measurements of the source-code are executed with a custom created Ruby-script. The differences between files is extracted with a diff-tool to context-format and parsed with a Ruby script. The differences and LOC of some files are checked manually to validate if the numbers are correct. The manual calculation matched the results from the automated script.

The implementation of the 'add-note' functionality is not identical in both techniques. This is because the techniques differ in architecture. This influences the measurements of the source-code.

8 Conclusion

Given an JEE application we want to add support for offline data manipulation. While applying this we want to reuse the most past of the original application. That is what the thesis is about. Migrating a JEE web-application to add offline functionality with high reusability.

For the research a JEE application was required. The company Sogyo where the thesis is created did not have an appropriate application that could be used. We have created a JEE web-application based on the way Sogyo creates web-applications. The JEE web-application is JDigiNotes and has its purpose to serve as the baseline for the migrations. It is a digital version of the yellow post-its to easily create notes. It allows logged-in users to create, update and delete notes. Notes from other can be seen as well. The JDigiNotes architecture has been compared with the PetStore from Sun to confirm the application is representative.

After the creation of the baseline we looked into various promising techniques offering some kind of client-side storage. During the investigation Gears, AIR and HTML 5 where suitable candidates for the research. The HTML 5 specification is recommended is 2010 or even later. Most browsers do not support this now. That why Gears from Google and AIR from Adobe where selected.

The offline functionality of viewing and updating notes is implemented during the migration. Offline means in this context: able to start the application/site and view the notes while being disconnected from the server for some reason. The updating of notes involves a synchronizing strategy. A manual (user triggered) strategy is chosen over background synchronization, because its less complex to implement. The synchronization is performed with Ajax. Both Gears and AIR have a different Ajax implementation.

The see differences in the reusability we measured the LOC before and after the migrations and see how the codebase changed along the way. Besides the LOC we looked into the change of architecture during the migration. The reuse is calculated from the baseline minus the changed and deleted lines. This divided by the baseline to have the outcome in percentage. The added lines do not influence the reuse, but do influence the growth. The results are shown in the table below:

Table 6: Aggregated results

Total	Baseline
Java	567
HTML / JSP	109
XML	83
JavaScript	-
Total	**759**

Gears	Reuse	Growth
586	99%	3%
129	77%	18%
123	100%	48%
224	-	-
1062	96%	40%

AIR	Reuse	Growth
657	98%	16%
168	33%	54%
119	100%	43%
277	-	-
1221	89%	61%

We see that the reuse of the AIR migration greatly differs from the Gears migration for the HTML/JSP. This happened because the AIR presentation-layer is created differently. The clients application is a distributable file that is partly created from the JSP's of the baseline. In Gears the same functional is achieved with more reuse and less extra code. The overall migration of AIR has a bigger growth than Gears. Unfortunately we could not use the same Ajax frameworks, because the AIR sandbox does not allow all JavaScript functions to work. This has a big impact source-code on the server-side. The implementations of the viewing of notes are not exactly the same due technical limitations that influences the results.

It is hard to predict if these numbers apply to other, even bigger JEE applications. The growth of the applications is depending on multiple factors like the size of the presentation-layer and used view technologies (html, or something else) that can be reused by either AIR or Gears. We have seen that the part of the

application that should be offline must have the domain objects involving that functionality. It implies that a part of the domain objects are duplicated on the client and saved in a database on the client. Not all properties (passwords, etc) are used of client-side domain objects. The effort to create this correlate to the amount and size of domain objects on the server used for the offline functionality.

The functionalities of Gears and AIR-api are exposed by JavaScript. The client-side uses these api's to perform the functionalities. All code manually created with JavaScript.

With both techniques it is possible to reach the goal of creating offline functionality. In this research it is not obvious which technique can be used best for reusability of the baseline, due some limitations and differences in implementations and used frameworks.

We recommend to try to apply these techniques to a larger scale application with for each migration the same server-side. And make use of the same Ajax frameworks. If Silverlight 3 from Microsoft is available it can also be used in the comparison as well as the HTML 5 specification.

9 References

Papers:

Braam, P. J. *"The Coda Distributed File System"*, Linux Journal nr50, 1998

Christodoulou, S. P. *"Evaluation of Hypermedia Application Development and Management Systems."* ACM Press, 1998, pp. 1 – 10.

Duhl, J. "*Rich Internet Applications"* white Paper, IDC analyse the future, 2003

Freeman, Eric and Freeman, Elisabeth *"Head First Design Patterns"*.
O'Reilly Media 2004. ISBN 0-596-00712-4

Kanter – Joel Kanter. *"Understanding Thin-Client/Server Computing"*. Microsoft Corporation 1999. ISBN 1-57231-744-2

Linaje, M. *"Engineering Rich Internet Application User Interfaces over Legacy Web Models"*.
IEEE Internet Computing 11(6): 53-59, 2007.

McConnell, S. "Code Complete Second Edition". Microsoft Press 2004. ISBN 0-7356-1967-0

Morales, R. *"MVC Web design patterns and Rich Internet Applications"*.
QUERCUS Software Engineering Group, 2007

ThroughLeadership. *"Improving User Experience with Rich Internet Applications"*
white paper, http://www.electronicink.com/thoughtleadership

Trerola, Rich. *"Beginning Adobe AIR: Building Applications for the Adobe Integrated Runtime"*, Wrox 2008

Internet Articles:

[AIR]
Online reference: http://www.adobe.com/products/air

[Dojo Offline]
Online reference: http://dojotoolkit.org/offline
Create offline web-application with Gears and Dojo: http://docs.google.com/View?docid=dhkhksk4_8gdp9gr

[DWR]
Online reference: http://www.directwebremoting.nl

[Gears]
Online reference: http://code.google.com/apis/gears

[Gecko Engine]
Online reference: http://developer.mozilla.org/en/Gecko

[Hibernate]
Online reference: http://www.hibernate.org

[HTML-5]
Online reference: http://dev.w3.org/HTML-5/offline-webapps

[HTML-4]
Online reference: http://www.w3.org/TR/html401

[HSQL-DB]
Hypersonic SQL database engine is a free to use 100% based on Java. It is a small and fast database engine compared to MySQL, PostgreSQL and others. HSQLDB is integrated in OpenOffice.org that has millions of users. Online reference: http://www.hsqldb.org

[Java Pet Store 2.0]
Online reference: https://blueprints.dev.java.net/petstore/

[JavaFX]
Online reference: http://www.sun.com/software/javafx

[JEE] refers to Sun - Java Enterprise Edition 5 based on Java Standard Edition 5. The previous version of the enterprise edition is called J2EE that refers to Java Standard Edition 1.4. JEE-5 has additional features: Enterprise Java Beans 3, Java Persistence API and can use the Java 1.5 language features.
Online reference: http://java.sun.com/javaee

[MVC - Sun]
http://java.sun.com/blueprints/patterns/MVC-detailed.html

[MooTools]
online reference: http://www.mootools.net

[Prism]
Online reference: http://labs.mozilla.com/2007/10/prism

[Silverlight]
Online reference: http://silverlight.net

[Sun - Servlet]
Online reference: http://java.sun.com/products/servlet

[Supported AJAX frameworks]
http://www.adobe.com/products/air/tools/ajax/community

[Sync Framework]
Online reference: http://msdn.microsoft.com/en-us/sync

Timóteo A. *"An approach to measure Java Code quality in an reuse environment."* 2008
http://worldofreuse.blogspot.com/2008/06/approach-to-measurement-java-code.html

10 Appendix

```
<web-app>
    <display-name>Notes Traditional Web-Application</display-name>

    <!-- Servlet's -->
    <servlet>
        <servlet-name>view note</servlet-name>
        <servlet-class>nl.sogyo.notes.web.ViewNotes</servlet-class>
    </servlet>
    ...
     <servlet>
        <servlet-name>logout</servlet-name>
        <servlet-class>nl.sogyo.notes.web.LogoutServlet</servlet-class>
    </servlet>

    <!-- Servlet-Mappings  -->
    <servlet-mapping>
        <servlet-name>view note</servlet-name>
        <url-pattern>/viewNotes</url-pattern>
    </servlet-mapping>
    ...
    <servlet-mapping>
        <servlet-name>logout</servlet-name>
        <url-pattern>/logout</url-pattern>
    </servlet-mapping>

    <!-- Error Pages -->
    <error-page>
        <exception-type>nl.sogyo.notes.web.UserNotLoggedOnException</exception-type>
        <location>/login</location>
    </error-page>
</web-app>
```

Figure 12: Deployment Descriptor: web.xml used for ServletContainer configuration

Figure 13: Part of update sequence with Gears (update_notes.js)

```
var lastUpdate = 0;

function getLatestUpdate() {
    try {
        var query = 'select note_creation_date from '+
                    'Note order by note_creation_date desc limit 1';
        var resultSet = db.execute(query);
        if (resultSet.isValidRow() && resultSet.field(0)) {
            return (resultSet.field(0));
        } else {
            return 0;
        }
    } catch(ex) {
        return 0;
    }
}

function requestNewNotes() {
    lastUpdate = getLatestUpdate();
    if (0 == lastUpdate) {
        NotesController.getAllNotes(updateNotescallBack);
    } else {
        NotesController.getNotesSince(lastUpdate, updateNotescallBack);
    }
}

function updateNotescallBack(notes) {
    for (var i = 0, len = notes.length; i < len; ++i) {
        var nid = notes[i].id;
        var nContent = notes[i].content;
        var nCreationDateMs = notes[i].creation.getTime();
        var nCreatorId = notes[i].creator.id;
        var nCreatorName = notes[i].creator.name;
        var nUpdateDate = notes[i].updated;

        if (nUpdateDate && nUpdateDate.getTime() > lastUpdate &&
            !isNewNoteLocally(nid)) {
            var nUpdateDateMs = nUpdateDate.getTime();
            updateRecord(note);
        } else {
            addRecord(note);
        }
    }
}
```

Figure 14: Part of update sequence with AIR (update_notes.js)

```
var lastUpdate = 0;

function lastUpdatedNote() {
    statement = new air.SQLStatement();
    statement.text ='select NOTE_CREATION_DATE ' +
                    'from Note order by NOTE_CREATION_DATE desc limit 1';
    statement.sqlConnection = conn;
    try {
        statement.execute();
    } catch (error) {
        console.log("QUERY error:" + error);
    }
    var result = statement.getResult();
    if (result.data) {
        return result.data[0].NOTE_CREATION_DATE;
    } else {
        return 0;
    }
}

function requestNewNotes() {
    lastUpdate = lastUpdatedNote();
    var request = new Request.JSON(
        {  url: 'http://localhost/new-notes',
           onComplete: updateNotesCallBack
        });
    if(0 == lastUpdate){
        request.send();
    } else {
        request.send('since=' + lastUpdate);
    }
}

function updateNotesCallBack(notes) {
    if (notes) {
        notes.each(function(note, index) {
            if (note.NOTE_UPDATED && note.NOTE_UPDATED > lastUpdate &&
                !isNewNoteLocally(note.NOTE_ID)) {
                updateNoteInLocalStore(note);
                updateNoteRowInTable(note);
            } else {
                addNoteToLocalStorage(note);
                addNoteRowToTable(note);
            }
        });
    }
}
```