Department of Computer Science

Erik Olsson Haglund

Fredrik Häggbom

Scrumptious

A Scrum Planning Tool Case Study to Evaluate the Rich Ajax Platform

Computer Science
C-level thesis (15hp)

Date/Term: 09-06-05
Supervisor: Donald F. Ross
Examiner: Martin Blom
Serial Number: C2009:03

Karlstads universitet 651 88 Karlstad
Tfn 054-700 10 00 Fax 054-700 14 60
Information@kau.se www.kau.se

Scrumptious:

A Scrum Planning Tool Case Study to Evaluate the Rich Ajax Platform

Erik Olsson Haglund

Fredrik Häggbom

This report is submitted in partial fulfillment of the requirements for the Bachelor's degree in Computer Science. All material in this report which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Erik Olsson Haglund

Fredrik Häggbom

Approved, 090605

Advisor: Donald F. Ross

Examiner: Martin Blom

Abstract

During the last year Tieto has been interested in evaluating the potential of a newly developed technology called Rich Ajax Platform (RAP). The RAP technology allows the developer to write Java code and deploy it on the web as a Rich Internet Application (RIA) with similar look and feel as a regular application ran in for example Microsoft Windows. The biggest benefit with a RIA is that it offers similar features to stand-alone applications but these features are deployed via a web server i.e. no installation at the client side is needed.

To evaluate this technique a Scrum handler prototype was developed using the RAP. Secondly, a comparison with another similar technique called Google Web Toolkit had been made. The Scrum process is an agile development method which is based on small iterations. The purpose of this study was to create a functional prototype in order to evaluate and demonstrate the RAP technology.

The result of the evaluation is that RAP is a strong competitor for developing Web 2.0 applications. Most of the concerns regarding RAP depend on the fact that it is a new technology and is still under development. Since Tieto in Karlstad currently develops RCP applications, the RAP technology is a logical successor since the developers do not have to learn a new technology.

Terminology

Abbreviations used in the dissertation:

AJAX – Asynchronous JavaScript and XML
GWT – Google Web Toolkit
JDBC – Java Database Connectivity
RAP – Rich Ajax Platform
RCP – Rich Client Platform
RIA – Rich Internet Application
SVN – Subversion
SWT – Standard Widget Toolkit
XML – eXtensible Markup Language

Contents

1 Introduction 1

1.1 Purpose 2

1.2 Subprojects 3

1.3 Existing systems 3

1.4 Summary 4

1.5 Dissertation Layout 5

2 Project Background 6

2.1 Scrum Process and Terminology 7

2.2 Scrum Team Organization 9

2.2.1 Example of a Scrum process

2.3 Scrum Process Requirements 14

2.4 Scrum Process – Automating 15

2.4.1 The Scrum prototype for different roles

2.5 Prototype System Setup 23

2.5.1 Eclipse
2.5.2 OSGi
2.5.3 Bundle
2.5.4 Equinox
2.5.5 Jetty
2.5.6 Java Database Connectivity (JDBC)
2.5.7 MySQL
2.5.8 Asynchronous JavaScript and XML (AJAX)
2.5.9 JavaScript
2.5.10 Qooxdoo
2.5.11 Widget Toolkit
2.5.12 Standard Widget Toolkit (SWT)
2.5.13 JFace
2.5.14 Rich Internet Application (RIA)
2.5.15 Subversion (SVN)

2.6 Rich Ajax Platform (RAP) 29

2.6.1 Background
2.6.2 Terminology

3 The Scrum prototype 34

3.1 Introduction 34

3.2 Drag and Drop 35

3.2.1 Background
3.2.2 Creating the widget

3.3 Implementation 41
3.3.1 Database design
3.3.2 Scrum Prototype System design

3.4 Summary 53

4 The RAP evaluation and comparison 54

4.1 GWT – an introduction 54

4.2 RAP vs. GWT 55
4.2.1 Similarities
4.2.2 Differences

4.3 The RAP/GWT Comparison - Conclusion 62
4.3.1 RAP
4.3.2 GWT
4.3.3 Summary

5 Conclusions and project evaluation 65

5.1 Evaluation of the technology 65

5.2 Evaluation of the project 66
5.2.1 Time management

5.3 Future work 69
5.3.1 Unimplemented features
5.3.2 The comparison

5.4 What we have learnt 70

References 71

A Details of the database tables 73

List of Figures

Figure 1-1 The main components of the Scrum prototype. 2
Figure 2-1 Scrum Application Overview....6
Figure 2-2 An example of a Scrum board in the middle of a sprint [44]....8
Figure 2-3 Scrum process overview [35]....10
Figure 2-4 Example of a product backlog....11
Figure 2-5 An example of a sprint backlog....12
Figure 2-6 Example of a burndown chart in the middle of a sprint 13
Figure 2-7 The virtual Scrum board 15
Figure 2-8 The login view....18
Figure 2-9 Overview of the main views and how they are accessed....19
Figure 2-10 Overview of the components used during the development. 23
Figure 2-11 Bundles in Equinox 25
Figure 2-12 Overview of JDBC 26
Figure 2-13 Screenshot of a RAP application....31
Figure 3-1 The drag and drop widget 35
Figure 3-2 Diagram of the database 42
Figure 3-3 Overview of the classes in the RAP application 43
Figure 3-4 UML diagram of the container classes 45
Figure 3-5 Illustration of the identity map 47
Figure 3-6 The view used to create a new product....51
Figure 3-7 Steps taken when a new product is created....52
Figure 3-8 Steps taken when a product is read from database. 53
Figure 4-1 Communication flow in a RAP application 56
Figure 4-2 Communication flow in a GWT application....57
Figure 4-3 Difference between RAP and GWT when loading lists 61
Figure 5-1 Distribution of time during the project 68

List of tables

Table 4-1 Time comparison between RAP and GWT. The unit used is seconds 59
Table 4-2 Comparison of transferred data when performing different commands.....59
Table 4-3 Advantages and disadvantages of the RAP technology.....62
Table 4-4 Advantages and disadvantages of the GWT technology 63
Table 5-1 Positive and negative aspects with this project 66

1 Introduction

During the last year Tieto has been interested in exploring the potential of a newly developed technology called Rich Ajax Platform (RAP). The RAP technology allows the developer to write Java code and deploy it on the web as a Rich Internet Application with similar look and feel as a regular application run in a web browser such as for example Microsoft Windows.

To evaluate this technique a Scrum handler prototype will be developed as a case study, using the RAP. A comparison to a similar technique called Google Web Toolkit will also be presented. This dissertation will cover background information about Scrum, RAP and some components used to develop a RAP application. A section describing the development of the Scrum handling tool will also be presented as well as a comparison between RAP and GWT.

The project is thus divided into two subtasks:

1. Create a web based application that will handle Scrum processes.
2. Examine and evaluate the potential of the RAP technique for building Rich Internet Applications.

1.1 Purpose

The purpose of this experiment is to evaluate and examine a technology called Rich Ajax Platform (RAP). In order to evaluate the technology we will develop a prototype tool to handle Scrum processes as a test case for RAP. RAP, which is a part of the Eclipse project [15], makes it possible to develop Rich Internet Applications by using only the Eclipse platform and the Java Application Programming Interface [22]. These terms will be explained in Chapter 2. An illustration of the relationship between user and database for the Scrum prototype is given in Figure 1-1.

The advantages of creating an Internet Application instead of creating a regular window application are platform independency and not having to install the application on every user machine. Creating a Rich Internet Application will give even greater advantages over a regular Internet Application. These advantages include less use of bandwidth by not having to reload the whole page when updates are needed and better user interactivity since a rich internet application is more like a regular window application than a web page. To sum up, RAP allows the creation of a web page with all the benefits of using the web but with most of the features that a regular application contains.

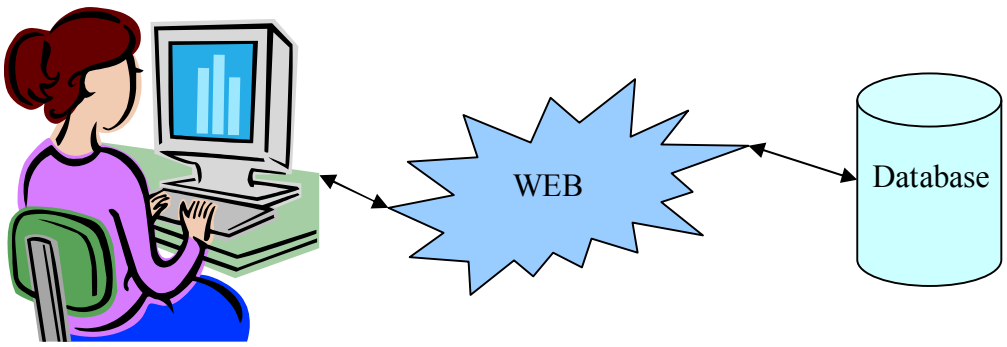

Figure 1-1 The main components of the Scrum prototype.

The Scrum prototype will be run by using a small operating system called Equinox which is an implementation of the OSGi [section 2.5.2] standard [18]. This implementation includes a web server called Jetty which will be used to deploy the Rich Internet Application. MySQL will be used to store the data. JDBC, which is a connector between Java and MySQL, will be used to access the database.

To evaluate the RAP technology, a comparison will be performed with the similar technology Google web toolkit (GWT) [21]. This comparison was chosen since GWT is the main competitor in this area and Tieto is interested in evaluating the difference between GWT and RAP.

1.2 Subprojects

This project has been divided into two sub-projects.

1. This sub-project is to create a web based application that will handle Scrum processes. The application will be developed using the technologies Eclipse, Java, RAP and MySQL [section 2.5.7].

2. The second sub-project will examine the RAP technique for building Rich Internet Applications. Tieto is interested in evaluating the potential of this new technique. This task will include a comparison with Google Web Toolkit [section 4.1] which is a similar technology for creating Rich Internet Applications.

1.3 Existing systems

Currently, Tieto uses two methods to manage their Scrum processes. The first method is a manual method i.e. writing the stories and tasks on a piece of paper. These papers are then placed on a pin board and the team moves them as the stories and tasks are under development or when they are done. This is a method that works well when the team is in the same place and if all the team members have access to the board at all time. For Tieto, this is often not the case. The team members are often located at different places, which makes it

very hard for all members to see the current status of the sprint, e.g. how many stories that are completed and how many stories they have left. This method is therefore not optimal for Tieto.

The second method that Tieto uses for their Scrum processes is an application called ScrumWorks [9]. This application, which is developed by a company called Danube [10], is a commercial application for handling Scrum processes. The reason why Tieto no longer wants to use this application is because it has two user interfaces, one on the web and one in a stand-alone application. The user interface on the web is very limited, and installing the stand-alone application on every computer is too time-consuming. Scrum Works also does not provide all the features that Tieto requires, e.g. prioritizing of tasks and the amount of time spent on a task.

1.4 Summary

Tieto wants to evaluate the technology Rich Ajax Platform. As part of this evaluation, a prototype Scrum process handling tool will be developed using this technology.

The experiment is divided into two subtasks:

1. Develop a Scrum process handling tool.
2. Evaluate the Rich Ajax Platform.

The experiment involves several different tools of which the main tools are:

- **Eclipse platform**: an IDE which will be used to handle the development, including writing, compiling, debugging and executing the application.
- **Rich Ajax Platform**: web enables Java code. Transforms some of the Java code to JavaScript which will be running at the client and the rest of the Java code will be executed at the server. This is the Rich Internet Application.
- **Equinox**: a small operating system which will run the server part of the Rich Internet Application.
- **MySQL**: a relation database which is used for storing the data.

These technologies are all open-source technologies which was an important consideration for Tieto when defining this experiment. The Rich Ajax Platform is a plug-in to the Eclipse Platform, which makes the choice of using Eclipse as IDE a natural choice. Eclipse has out of the box support for Equinox, which also makes this a natural choice.

1.5 Dissertation Layout

The dissertation is divided into five chapters:

- Chapter 1: An introduction to the project.
- Chapter 2: Background and description of Scrum, RAP and all other components used in the project.
- Chapter 3: Description of the Scrum process handling tool, and how it was developed.
- Chapter 4: Evaluation of the RAP and a comparison between RAP and GWT.
- Chapter 5: Conclusions and evaluation of the project.

2 Project Background

The background to the Scrum prototype is presented in this chapter. Recall that the first part of the project is a prototype implementation of a Scrum application which will be used to illustrate and evaluate the RAP technology. Essentially, the Scrum application consists of a user web interface to a database, via the web (*Figure 2-1*). The background to the Scrum process and how this translates to the Scrum application is presented in section 2.1.

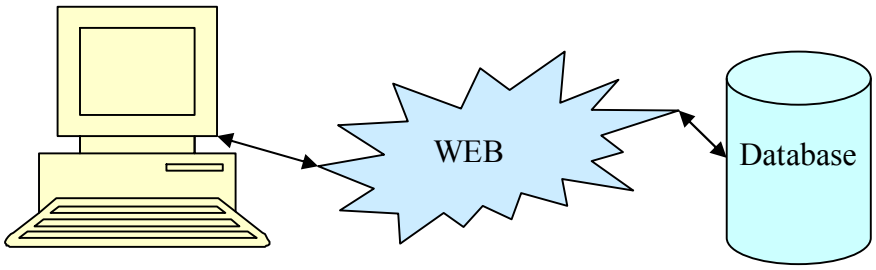

Figure 2-1 Scrum Application Overview

Then in section 2.5 an overview of the prototype setup using RAP technology (*Figure 2-10*) is described. In order to see the connection between the Scrum application and the RAP technology, *Figure 2-10* shows the details of the RAP technology and components used. The components used in the client-server division can also be seen in detail.

In section 2.6, a more general background to the RAP technology is presented.

2.1 Scrum Process and Terminology

Scrum [24] is a process which can be used in a software development project. The process, which is based on iterations, was presented by Schwaber and Sutherland in 1995 [13]. They both came up with the same idea independently of each other in the early 1990's. When they discovered that they had similar ideas, they teamed up and wrote a paper describing Scrum [12].

In Scrum the iterations are called **sprints**. A sprint is a pre-defined number of weeks in which the team will complete their user stories [25] and tasks. Typically a sprint lasts 2-4 weeks. A ***user story*** is a specification of the product that is being developed, and is formulated as one or two sentences in non-technical terms. The development team breaks down each story into several ***tasks***. These tasks are more concrete descriptions of what is to be done.

Each sprint begins with a ***sprint planning***, where the team chooses stories from the ***product backlog***. The product backlog is a set of features for the software being developed. The set is ordered by ***priority***. The team also estimates how much ***time*** each story will take. This is done by estimating the number of ***story points*** each story will take. A story point is a time unit and can be defined in any way the team requires. There are a number of ways to estimate a story point and one example of a common technique is "planning poker" [45]. In this estimation technique, the team members get a deck of cards which contains 12 cards marked with the numbers 0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 100 and "infinity". These numbers represents story points. For each of the stories all team members individually estimates the number of story points for the current story by choosing a card. The cards are not shown to the other team members until all team members have made their choice. If all team members chose the same number of story points, this will be their estimation for the task. If the team members did not choose the same number of story points, they will have a small discussion were they motivates their choice, and then revote until they all choose the same number of story points. However, instead of using story points on stories, Tieto uses two attributes called ***benefit*** and ***penalty*** to prioritize the stories. These attributes are numbers representing the

benefit of having the feature respectively the penalty of not having the feature. If a feature has a high numbers of both benefit and penalty, it is a highly prioritized feature and vice versa. Tieto uses points on tasks, instead of stories.

As they choose stories, the team members put them in a ***sprint backlog***, which at the end of the sprint planning is a subset of the product backlog. The sprint backlog contains the stories the team will develop in the current sprint. When the stories are chosen, the team breaks them down into tasks.

When the team has developed all tasks for a story, the story will be marked as done, and the Scrum Master will update a chart called ***burndown chart***. This chart shows whether the team is on schedule according to their story point estimation at the sprint planning.

Both the sprint backlog and the burndown chart are often placed on a ***Scrum board*** (Figure 2-2).

Figure 2-2 An example of a Scrum board in the middle of a sprint [44].

Stories and tasks are first located in the "***not checked out***" column. When someone starts working on a task, he or she moves it to the "***checked out***" column. This is a simple way to show the other team members that a task is under development. When the team member finished developing the task, he or she moves it to the ***done*** column. When all the tasks are moved to "done" the whole story is done. In Figure 2-2, the white notes are stories and the yellow notes are tasks.

The burndown chart is updated at the ***daily stand up meeting*** which is a short meeting (approx 15 min.) held every day. Every member in the team gets to explain what they have done the past day and what they will do next.

If the Product Owner comes up with new stories in the middle of the sprint, they will be placed under the ***unplanned items*** column. If there are some time left at the end of the sprint, team members can consider these stories.

Under the ***next*** column there are stories that have been placed there under the sprint planning meeting. The team does not commit on doing the stories this sprint, they only consider them if they have some time left after all the other stories are done.

2.2 Scrum Team Organization

In Scrum there are three main roles: ***product owner***, ***scrum-master*** and the ***team***.

1. The ***product owner*** is the person who speaks for the customer. The product owner writes the user stories and adds them to the product backlog.
2. The ***Scrum-master's*** main task is to remove any obstacles that prevent the team from achieving the sprint goal. Since Scrum does not have any hierarchy with a leader in charge, the Scrum-master is not the leader of the team. The Scrum Master also keeps the Scrum board updated for example by updating the burndown chart when stories are done and by creating the story notes when planning a sprint.
3. The ***team*** is a group of people which has the responsibility for delivering the product. The recommended number of people in a team is 5-9.

These three roles are members of a category called "***pigs***", which in a Scrum process are the people who is working on a product. There is also a category called "***chickens"*** which includes people that have a "view only role" for the products. This is often the customers of the product or the users that the product is being developed for. When having a view only role, they can see how the Scrum process develops, but they can not change or update anything hence the view only role.

At the end of a sprint there is a ***demonstration*** of the stories implemented in the current sprint. Present at the demonstration is the product owner, the Scrum Master and the developing team. The product owner can criticize how a story turned out in the application. He can also decide to delete other stories from the product backlog, since they might not be needed anymore.

After the demonstration there is a ***retrospective*** which includes the developing team and the Scrum Master. In the retrospective meeting, which is an evaluation of the past sprint, the following are examples of commonly topics discussed:

- What did we do well?
- What could we have done better?

A basic Scrum process is shown in Figure 2-3.

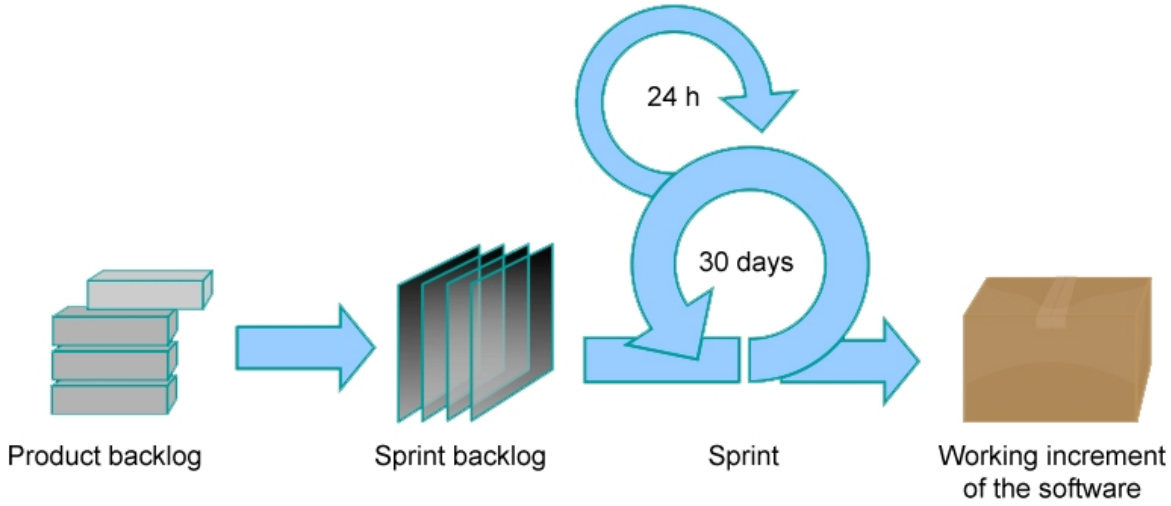

Figure 2-3 Scrum process overview [35]

2.2.1 Example of a Scrum process

In this section, an example of a Scrum process will be presented. This process will be used throughout this dissertation to give a better description of the functions in the prototype and the methods used to create the prototype.

In the Scrum process, an application that handles booking of rooms will be created. The application will offer features for teachers to book a room for lectures. The application will also offer features such as the ability to order catering with the room booking as well as other equipment (e.g. overhead or projector). When a teacher reserves a room, the teacher can add participants to a booking e.g. if a teacher books a room for a lecture, he or she is able to specify which students may attend the lecture.

The specification of the application is created by the customer and the product backlog will be created by the product owner. The product backlog for this project can be seen in Figure 2-4.

Item #	Priority	Description
1	10	Book a room
2	17	See available rooms at given time
3	25	Administrator window to handle bookings
4	40	Administrator window to handle users
5	50	Add participants to a booking
6	59	Manage catering details
7	65	Order catering to a room at given time
8	78	Administrator window to handle rooms
9	85	Administrator window to handle catering

Figure 2-4 Example of a product backlog

At the first sprint planning, the team estimates how many of these items they will complete during the first sprint. This is done by estimating how many story points each item will take. In this project a story point is defined as one day of work for one person. The team in this project consists of six persons and a sprint is 10 days which gives a total of 60 story points per sprint (6 persons * 10 days). The team chooses one item at a time from the product backlog. They estimate the story points for these items and the Scrum Master puts them in the sprint backlog as stories. This is done until the total number of story points in the sprint backlog reaches 60, i.e. when the team has enough stories for the current sprint.

In the next step of the sprint planning, the team breaks down each story into several tasks.

In this project, the team chooses stories and breaks them down to tasks according to the first sprint backlog, which can be seen in Figure 2-5.

Story id	Story/task	Story points
1	**Book a room**	**25**
	- Create project, set up environment	
	- Create database	
	- Create GUI for selection of room and time	
	- Save selection of room and time in database	
2	**See available rooms at given time**	**15**
	- Read room bookings from database	
	- Populate table with the bookings read from database	
3	**Administrator window to handle bookings**	**20**
	- Create new form	
	- Read bookings from database and show bookings to user	
	- Add buttons to delete or modify bookings	
	- Update database according to the administrators modifications	

Figure 2-5 An example of a sprint backlog

When the team starts developing the room booking application, the members choose tasks from the sprint backlog and mark them as "checked out", which basically means that some team members are working on that task. When the task is finished it is marked as done. When all tasks for a story are marked as done, the team marks the whole story as done, and the Scrum Master updates the burndown chart. Figure 2-6 shows the burndown chart for this sprint when the two first stories are done. The blue line in the chart is where the team should be when on schedule and the red line shows if the team is on, behind or ahead of schedule. When the first story is done (second red dot) the team are a bit ahead of schedule which means that the story did not take as many story points as the team estimated. For the second story, it is the other way around; the story took more story points then the team estimated, which makes the team behind schedule after this story. The Scrum prototype in this experiment will help the team with the Scrum planning. For example, the users will be able to create projects and bind the users in the team to the project, create a backlog for the project and create sprints for the project. Each user can commit to a story or task, which means that the user takes responsibility for finishing the task. A burndown chart will also be generated automatically as the users complete their stories and tasks.

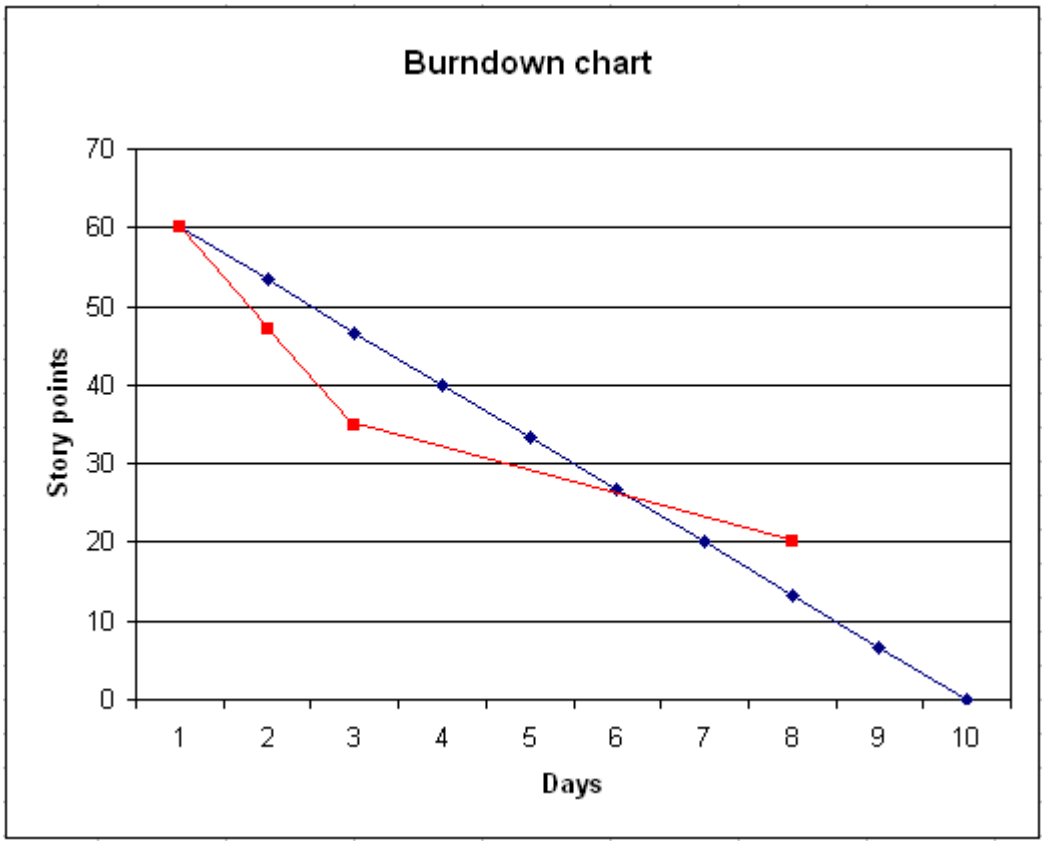

Figure 2-6 Example of a burndown chart in the middle of a sprint

After the sprint is finished, it is time to show a demo of the room booking application and the stories implemented in this sprint. Present at the demo are the developing team, Scrum Master and the product owner. In this demo, the product owner is satisfied because the stories implemented equal his expectations. He also decides to lower the priority of the add participants story from the product backlog. He did this because he simply does not see the use of this feature in the near future and would rather have the catering feature implemented before that.

After the demonstration it is time for a retrospective. This is an evaluation of the current sprint. Only the Scrum Master and the developing team will be present. In the retrospective meetings, the team members write down positive and negative situations during the sprint. If any negative points arise, for example if there were any misunderstandings in the team that caused the team to fall behind schedule at the end of the sprint, these have to be solved to avoid the same issues repeating themselves during a future sprint.

2.3 Scrum Process Requirements

To sum up, the required components of a Scrum process are:

- **product backlog** – A list of all features specified for the product
- **sprint backlog** – A list of features estimated to be done in the sprint
- **stories** – The features in the sprint backlog
- **tasks** – A subpart of a story

The Scrum prototype has to contain all these components. To keep track of a sprint and which stories that are done, checked out or not checked out, a Scrum board is used. This Scrum board will also exist in the Scrum prototype, and will serve the same purpose as in the example above, i.e. show the status of the task for the current sprint and show a burndown chart. Figure 2-7 shows the virtual Scrum board created in this experiment. The left list is the product backlog, which means that it lists all stories for the product. For each story, it also lists the tasks connected to the story, with an icon indicating if the story is done, checked out or not checked out (green, yellow or gray icon).

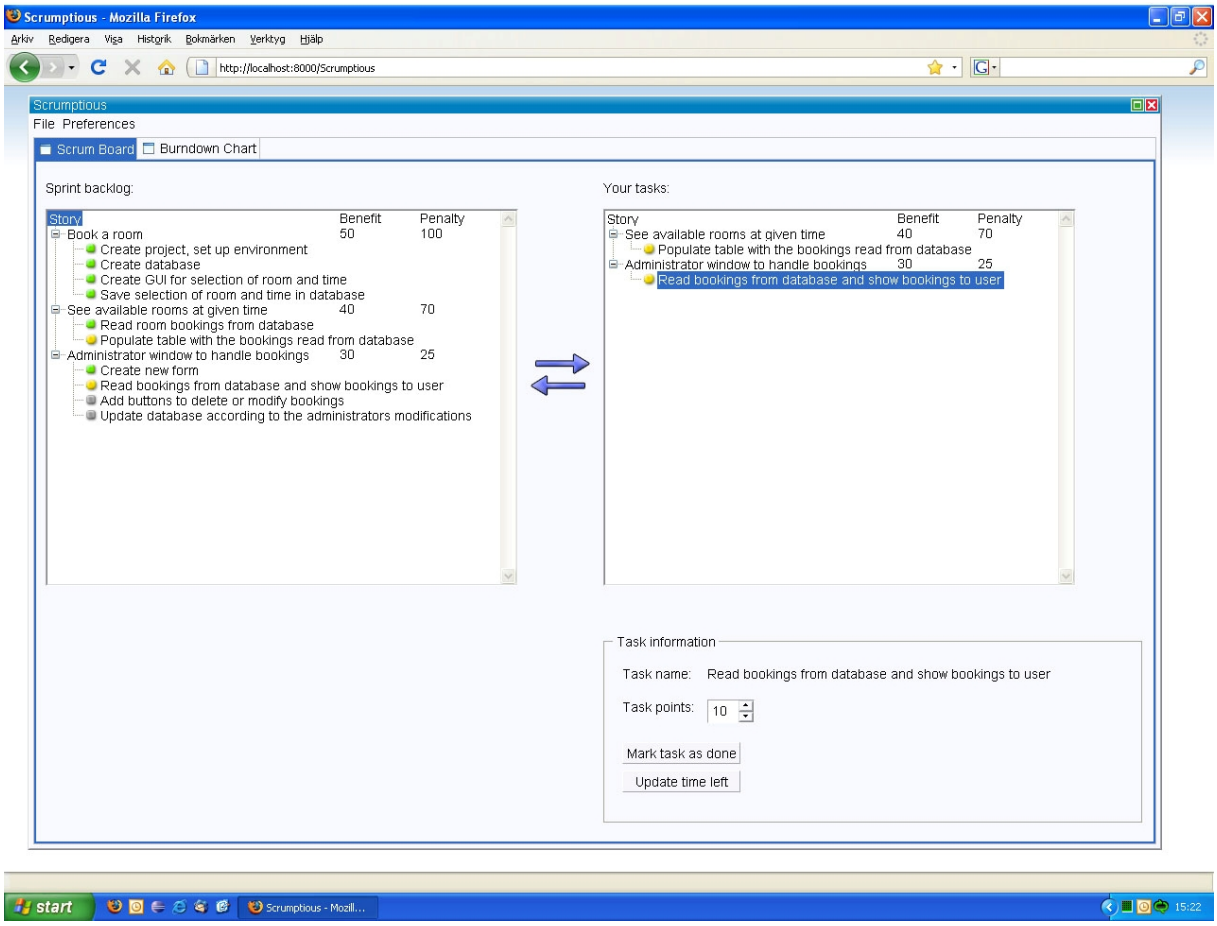

Figure 2-7 The virtual Scrum board

The Scrum prototype also has to be able to handle different kind of Scrum roles since the different roles have different tasks in a Scrum process. For example, if a team member logs in, he or she should not be able to alter the product backlog, whilst a product owner should have the authority to alter the product backlog.

The Scrum prototype has to have support for handling more than one product for a user. For example, a user can be a team member in one project, and a Scrum Master in another. The Scrum prototype has to be able to handle this situation. However in reality participating in two or more projects at the same time is rare.

2.4 Scrum Process – Automating

In this experiment a prototype to automate the Scrum process in the example [section 2.2.1] will be developed. The information about the components handled in the process, e.g.

product, sprints, stories, tasks and users will be stored in a database and a user interface implemented to handle the information. Since the different user roles in Scrum have different authorizations and commitments, the user interface will be different depending on the logged in user's role. For example, if a team member logs in, he or she should not be able to create sprints or edit the product backlog.

In the Scrum prototype these are the existing roles with their corresponding tasks.

1. **System administrator**
    - Performs system configurations and set-up.
    - Backup of data
    - IT-support

2. **Administrator**
    - Performs management of projects:
        - Create
        - Update
        - Close
        - Delete
    - Add and remove of users
    - Assign access rights to users (at least Scrum Master role)
    - Can see all projects

3. **Product owner**
    - Maintains the product backlog
    - Import backlogs from spreadsheet

4. **Team member**
    - Updates work items from Sprint list
    - Export backlogs as spreadsheet
    - Add tasks to backlog items (in sprint backlog and product backlog)

5. **Scrum Master**
    - Maintains the Sprint plan

- Maintains sprint backlogs
- Add and remove users to project
- Assign roles to users
- Import backlogs from spreadsheet

6. **Chickens**
    - Has a view only role

2.4.1 The Scrum prototype for different roles

When the users logs in using the view shown in *Figure 2-8*, they will be presented with different views depending on what system role they have. However if the user does not have a system role, i.e. the user is not an administrator or system administrator, they will be presented with a view showing all products they are a member of, and which role they have in the products. The user can then select a product and will be presented with some view, depending on which role the user has in the product. For an example of the team member view, see *Figure 2-13* and for an example of the view used to create a new product, see *Figure 3-6*.

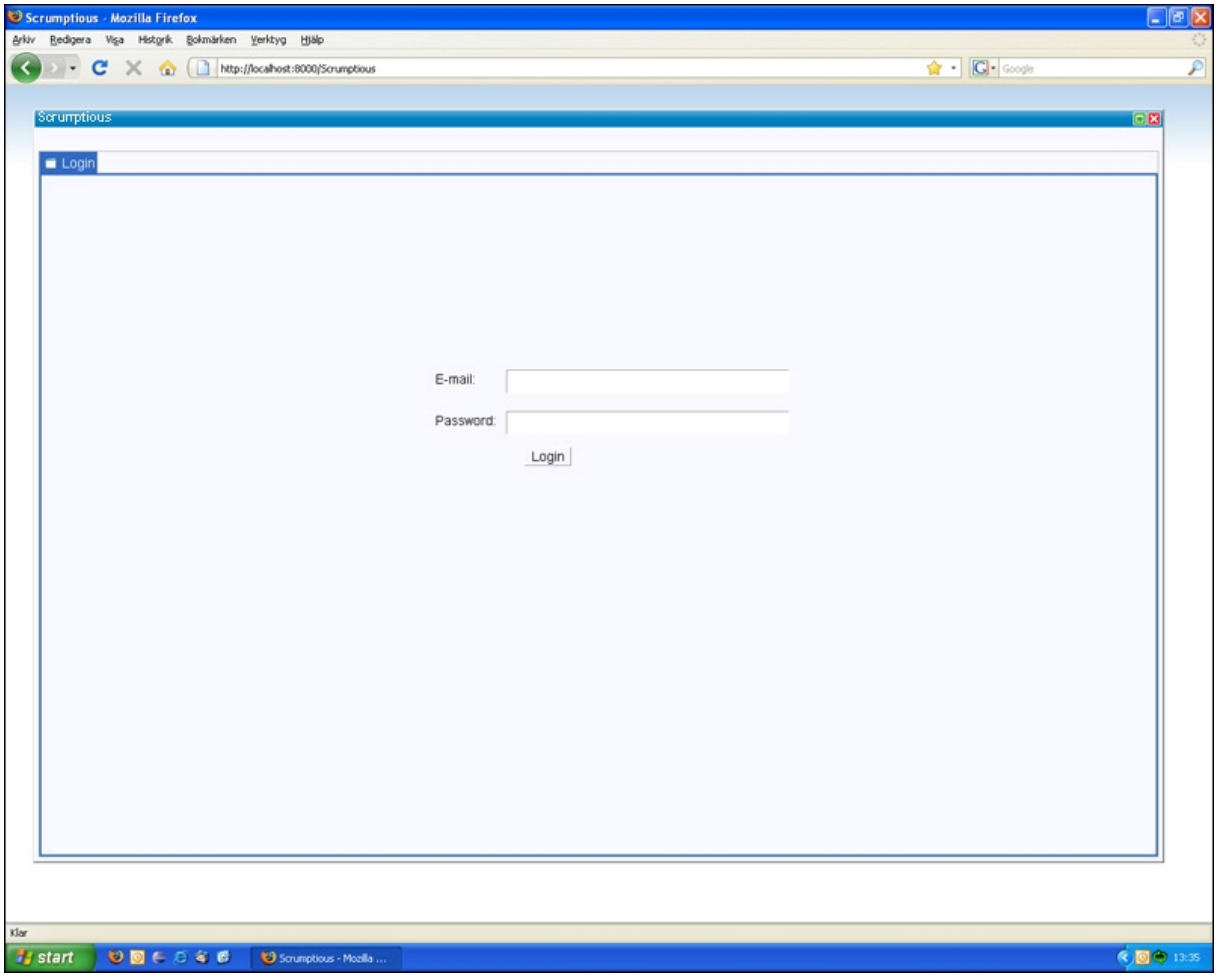

Figure 2-8 The login view

Figure 2-9 shows a scheme of the main views and which of the roles who can access them.

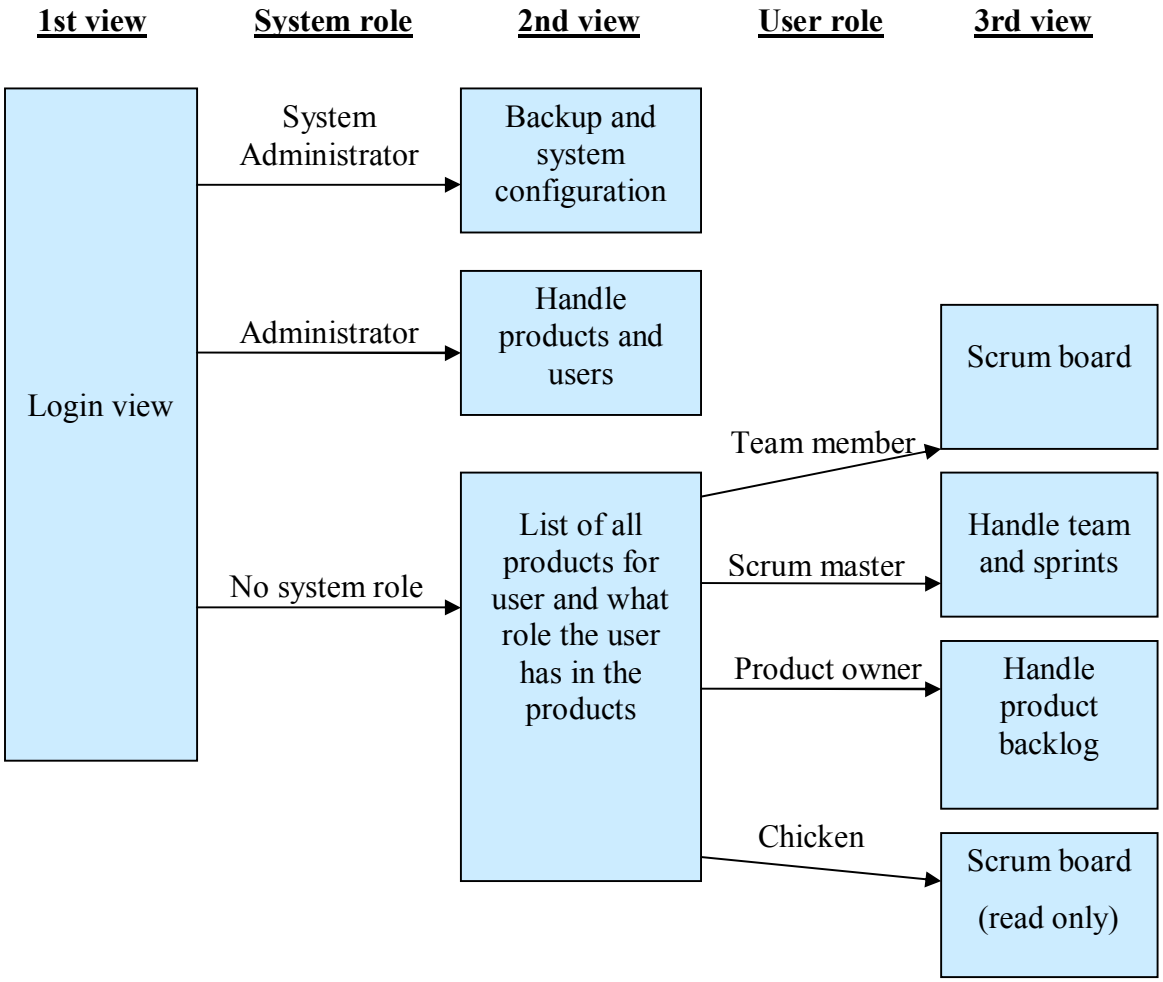

Figure 2-9 Overview of the main views and how they are accessed

System administrator

A logged in system administrator is prompted with two views:

1. Dump of the identity map
2. Backup/restore

The first view shows a detailed real-time dump of the identity map [section 3.3.2.2]; containing information about which products with corresponding sprints, stories and tasks that are located in the memory at the server.

The other view that the system administrator has access to is for backup and restoration purposes. The functionality to export a dump of the database as well as restoring from a previous backup can be found in this view.

Administrator

When an administrator logs in, he or she will be directed to a perspective with two tabs (two different views): Handle Products and Handle Users. In the first tab, Handle Products, the administrator can see a view with a table showing all products from the database. The user can choose to add a new product, which will show a dialog box where the administrator enters the name of the product and selects a Scrum Master and product owner. He or she can also delete a product and close a product, i.e. set the status of a product to done. If the administrator selects a product in the table, he or she can change Scrum Master and/or product owner of the product. This is done by showing a dialog box with two lists containing all team members of a product. The administrator selects a Scrum Master in the first list, and a product owner in the second list, and saves the changes.

In the second tab of this perspective, Handle Users, a table showing the name, e-mail address and system role of all users are shown. The administrator can add a user by filling in name, e-mail and system role in a dialog box. If the new user is not an administrator or system administrator, the system role can be set as “none”. The administrator can also delete a user, by selecting it in the table and pressing a delete button. By selecting a user in the table and pressing an update button, the details of a user can be modified.

Product owner

As described in the Scrum example in section 2.2.1, the product owner handles the product backlog. When a product owner logs in, he or she is shown an overview of the product backlog in form of a table consisting of three columns: Story name, benefit and penalty. The table shows all stories for the product. The product owner has the ability to add a new story to the backlog, delete a story from the backlog or edit an existing story. When adding a story to the product backlog, a dialog box will pop up. In this dialog box there are four text fields; story name, description, benefit and penalty. The product owner must enter values in all fields to add a story.

When the product owner wants to edit a story, he or she selects a story in the table and presses an Edit story button. The same dialog box as when creating a new story appears, with the difference that the text fields are already filled in with the values from the selected story. The product owner can edit the text fields, and then press an Ok button to save the changes.

If the product owner wants to delete a story, he or she simply selects a story in the table and then presses a delete button.

Team member

When a team member logs in he or she is prompted with a virtual version of the Scrum board (see Figure 2-7 for a screen shot).

The user can see the current sprint backlog with three states for every task:

1. Done
2. Checked out (i.e. someone is working on it)
3. Not checked out (i.e. someone can start working on the story)

The Scrum board will contain the drag and drop widget [see section 3.2 for further details], which means that there will be two trees in the view. In the left tree, all stories and tasks for the current sprint will be shown, including the states showing if a task is done, checked out or not checked out. In the right tree, the logged in user can see all tasks he or she have committed to complete and which story they belong to.

A user commits to a task by dragging a not checked out task from the left tree over to the right tree and then pressing commit.

The user can also add tasks to a story and remove tasks from a story in this view. This is done by right-clicking and selects either “Add new task” or “Remove task” in the context menu [49].

When the user has committed to a task, he or she can select the task in the right tree, and see information and perform operations on the task in a group box below the tree. The user can see the full name and the total number of story points for the task. He or she can also mark the task as completed or set number of points left until the task is completed. This is useful if a task is large and has a high number of points, since the task would go from nothing to done if the user can not update amount of time worked on the task during the development.

There is also a burndown chart visible for the team member. The chart is updated every time a new task or a whole story is reported as done.

Scrum Master

The Scrum Master's main task in this Scrum prototype is to handle the sprints and the sprint planning. When a Scrum Master logs in, he will be presented with a view consisting of two tables: one for all the stories in the product (i.e. the product backlog) and one table for all the sprints in the product. The sprint table will show both current sprints and sprints that are done as well as upcoming sprints.

In the view there are two buttons: one for adding a new sprint and one for modifying an existing sprint. When the Scrum Master chooses to add a sprint to the product, he or she will be presented with a dialog box with two lists and two text fields. One of the lists is populated with the stories associated with the product (i.e. the product backlog) and the other list is empty. The Scrum Master may drag and drop stories associated with the new sprint from the product backlog list into the empty list. He or she then enters start date and end date in the text fields and saves the new sprint by pressing an Ok button.

The Scrum Master can also choose to modify a sprint. He or she will then be presented with a view containing a list which shows all stories in the selected sprint. The Scrum Master can add tasks to a story by right clicking a story and choose "Add task" in the context menu. A dialog box will pop up and the Scrum Master must enter a name for the task to create it.

Chicken

The chicken role will only allow the users to view status of a current project. When a chicken logs in, he or she will be directed to the virtual Scrum board, but in contrast to the team members' virtual Scrum board, the chickens can not operate on it i.e. they can not commit to stories nor mark any stories as done. However, the chicken can see all stories and tasks with corresponding status, which allows the chicken (e.g. a customer) to follow the process of the product. The chickens can also see the burndown chart, which is the same view as the team members have.

2.5 Prototype System Setup

This section contains descriptions of all the components used when creating the Scrum planning prototype.

Figure 2-10 presents an overview of the components and how they relate to each other during the development. When deployed, the Eclipse component is changed to a web server, e.g. Tomcat [59].

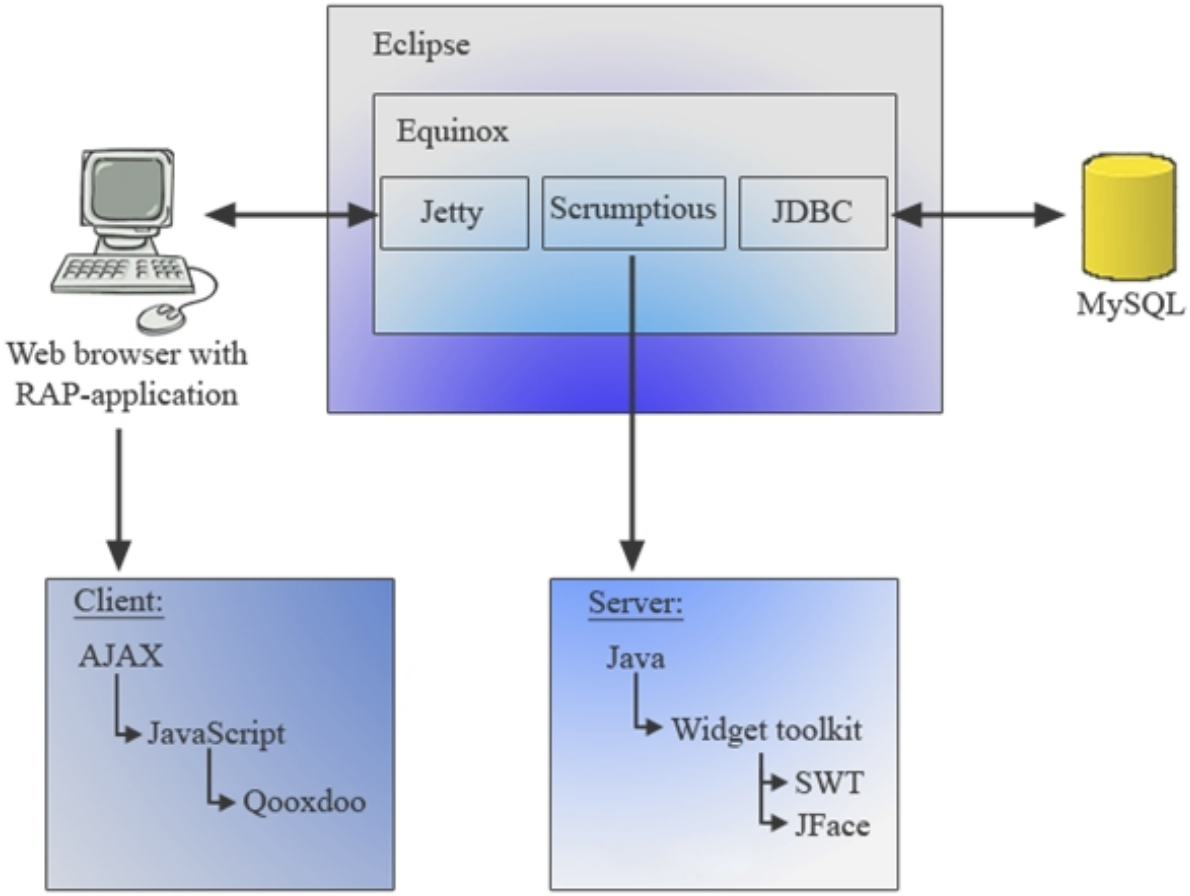

Figure 2-10 Overview of the components used during the development.

2.5.1 Eclipse

Eclipse [15] is an open source multi-language software development platform which includes a source code editor, compiler and a debugger. Eclipse also supports the use of plug-ins to extend its features. When the first version of Eclipse was released, the application was

aimed at Java developers since it included Java Development Tools (JDT) [39]. In the current version, Eclipse can also be used with many other programming languages, such as C++, C and Python. Eclipse is currently developed and maintained by the Eclipse Foundation [1], which is a non-profit foundation that develops open source applications.

2.5.2 OSGi

OSGi is an open standard for an operating system that defines a dynamic module system for Java. When this standard is implemented, the resulting implementation is a small operating system. The OSGi technology provides functionality that allows applications to be constructed from small, reusable components called bundles [section 2.5.3].

OSGi makes it possible to alter the composition of bundles dynamically without requiring a system restart. OSGi also makes it possible to restart a single bundle without interfering with other bundles. This is possible by using an architecture which provides a feature to a bundle which allows the bundle to discover other new or updated bundles under run-time without the need to restart the application [18]. Equinox [section 2.5.4] is the implementation of the operating system defined by the OSGi standard used in this project.

The certification of the OSGi standard is handled by the OSGi Alliance which formerly was known as the Open Services Gateway Initiative, but this is now an obsolete name.

2.5.3 Bundle

A bundle is an archive containing resources which may be Java class files or other data such as HTML files. The archive also contains a manifest, which is a file that provides descriptions of the bundle and the resources. One advantage of the manifest is that the developer can declare which functions will be accessible and to which bundles. For example, a function in bundle A can be declared to be accessible by bundle B, but not by bundle C. The developer can also declare on which platform the bundle can be run, which means that a bundle can be started if the operating system is UNIX and another if the operating system is Windows. [17]

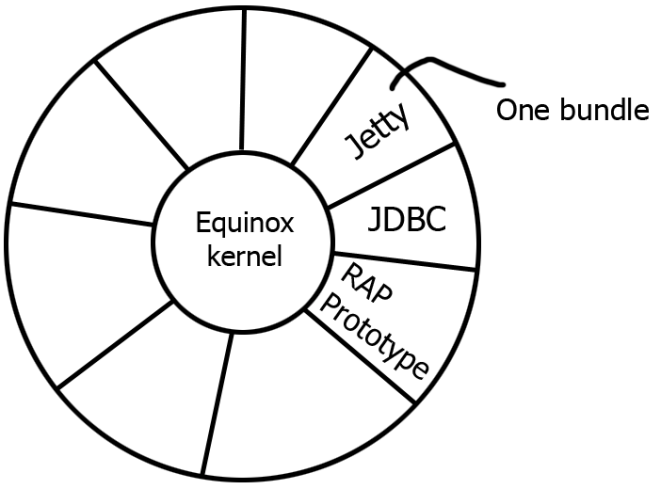

Figure 2-11 Bundles in Equinox

Figure 2-11 shows the relationship of bundles to the Equinox kernel.

2.5.4 Equinox

Equinox [11] provides an OSGi certified implementation of the OSGi standard [section 2.5.2] and is therefore a small operating system. The Equinox operating system has been developed as an Eclipse project and can therefore be launched from within Eclipse. We will use this implementation when launching our Scrum prototype. The Scrum prototype will be launched as a bundle in Equinox.

2.5.5 Jetty

Jetty [14] is an open source web server implemented in Java. Jetty is, for a web server, very small in size which makes it possible to run it as a bundle in an OSGi implementation. Jetty is part of the Eclipse foundation [1].

2.5.6 Java Database Connectivity (JDBC)

Java Database Connectivity (JDBC) [20] is the link between the client and the database and provides methods for get and set data from a database. The connections between the java application, JDBC and the database are shown in Figure 2-12.

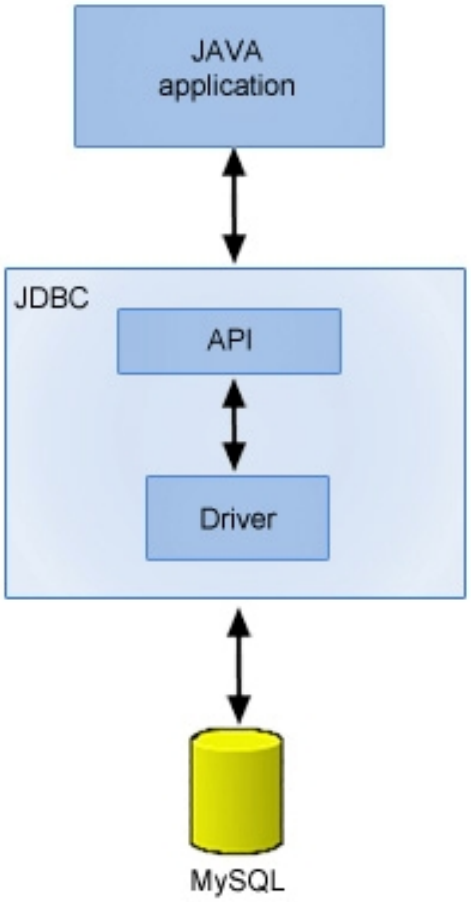

Figure 2-12 Overview of JDBC

2.5.7 MySQL

MySQL [33] is one of the most common relational database management systems. The system is open source. MySQL can be used on many different platforms, including the most common operating systems Linux, Microsoft Windows and Mac OS X. As the name indicates, the queries in MySQL are written in the SQL [5] language (Structured Query Language), which is a standardized language for querying and modifying data and managing databases.

2.5.8 Asynchronous JavaScript and XML (AJAX)

Asynchronous JavaScript and XML (AJAX) is a technique for developing Rich Internet Applications. XML [2] is an abbreviation for eXtensible Markup Language and is a method for data structuring. XML provides rules for creating text-based data storage which are unambiguous and platform independent. When using AJAX, the web browser can retrieve data from the server without having to reload the page like regular Internet Applications. The main advantages of using AJAX are that less bandwidth is consumed when an update is needed and more interactive internet applications may be provided. This is because all the data on a specific homepage does not have to be reloaded every time a user does a HTTP request. The improved interactivity comes from the fact that the page can reload a specific part of the homepage when the user enters a specific input, instead of requesting a whole new page. This is possible because the client sends and retrieves XML files instead of regular HTTP request/response [3]. The disadvantage of AJAX is that although it uses less bandwidth when updating a page, it communicates more often with the server. This means that less bandwidth is used, but the bandwidth is used more often compared to a static HTML page.

2.5.9 JavaScript

JavaScript [6] is a scripting language which is very commonly used to write web pages. The language is used to extend features for the web pages (e.g. menu, ticking clock). It is often used to control and dynamically alter a web page depending on the user's actions. Despite the name and that it originally was designed to look like Java, it is essentially unrelated to the Java programming language. JavaScript was introduced 1995 in the, at the time, popular Netscape Navigator browser (1995-1999).

2.5.10 Qooxdoo

Qooxdoo [16] is an open source Ajax web application framework and is an object oriented version of JavaScript. RAP uses the Qooxdoo framework when compiling the web application i.e. the Java code is converted to Qooxdoo JavaScript. It is possible to extend the RAP

widgets by creating custom widget using Qooxdoo. In this experiment this has been done to implement a Drag and Drop feature.

2.5.11 Widget Toolkit

Widget [4] is short for window gadget. A graphical user interface is based on one or more widgets. Some examples of widgets are buttons, textboxes and list boxes. A widget toolkit is a set of widgets.

2.5.12 Standard Widget Toolkit (SWT)

The Standard Widget Toolkit (SWT) is a graphical widget toolkit used within the Java platform. To display a widget, SWT accesses the operating system native GUI libraries.

2.5.13 JFace

JFace is a toolkit that provides helper classes for developing User Interface features. It acts as the layer above the widgets. JFace allow features as populating, sorting, filtering and updating widgets. It also allows the programmer to assign a created feature to a specific behaviour e.g. push a button.

2.5.14 Rich Internet Application (RIA)

Rich Internet Applications (RIA) [19] are applications that have some characteristics (look and basic functions) of a regular desktop application, but are launched and used in a web browser. The Rich Internet Applications are launched through a framework which displays the application in the users' web browser. These frameworks extend the functionality of a regular web page by providing methods to the developer. One example of a RIA framework is Adobe Flash [31]. To view the applications, the user often has to install the RIA framework software on the computer, for example to view an RIA in Adobe Flash, the user must have Adobe Flash player installed.

The main advantage for these applications is that it gives the web more features (or rich features, hence the name).

2.5.15 Subversion (SVN)

Subversion (abbreviated as SVN) [34] is a version control system that handles the developers' source code either inside Eclipse or directly in the operating systems. SVN is compatible with many IDE's [7], often by using a plug-in. We will use SVN from within Eclipse. SVN is used to maintain and ease distribution of files. Developers often use this for their source code. SVNs main features are that it can merge files automatically and has a full history of all files in the project. The developers may check out a file, alter the file in some way, commit it and SVN will automatically merge the file with the one in the repository. When the file is committed and merged, it can also be reverted to a previous version, i.e. undo the changes [8].

2.6 Rich Ajax Platform (RAP)

Rich Ajax Platform (RAP) [40] is a technology to develop Rich Internet Applications using Java libraries, Qooxdoo [section 2.5.10] and Eclipse [section 2.5.1]. RAP provides implementations of SWT [section 2.5.12], JFace [section 2.5.13] and the Eclipse Workbench [23] each of which has been web enabled (i.e. converted to work on the web).

2.6.1 Background

Over the last few years, Rich Internet Applications [section 2.5.14] have become more common and the number of technologies used to create these applications is on the increase. The RAP technology was founded in March 2006, and became an Eclipse project in June 2006. The team that develops RAP decided that they would use an already existing technology to create regular window applications, called Rich Client Platform (RCP) [27].

A description of an RCP application is an application that performs most of the calculations at the client instead of doing them at a server and just showing the results at the client. An example of an RCP application is Vuze (formerly Azureus) [42] which is a rich featured bit torrent [43] client.

The Eclipse team used the main structure and components from RCP and converted the user interface components to work on the web.

The RAP technology is still under development, since not all GUI [28] components and related methods are implemented in the RAP API [22] as of yet.

2.6.2 Terminology

RAP has some classes that have to be implemented and some classes that are predefined and which can be used to help the developer. The main components of a RAP application are listed in this section. Figure 2-13 shows a screenshot of a RAP application and will be used in this section to help describing some of the components of RAP. In the screenshot, the number represents the following:

1. Perspective
2. Menu bar
3. View
4. Widgets
5. Tabs, used to choose view

Perspective, view and menu bar are components that will be described later in this section.

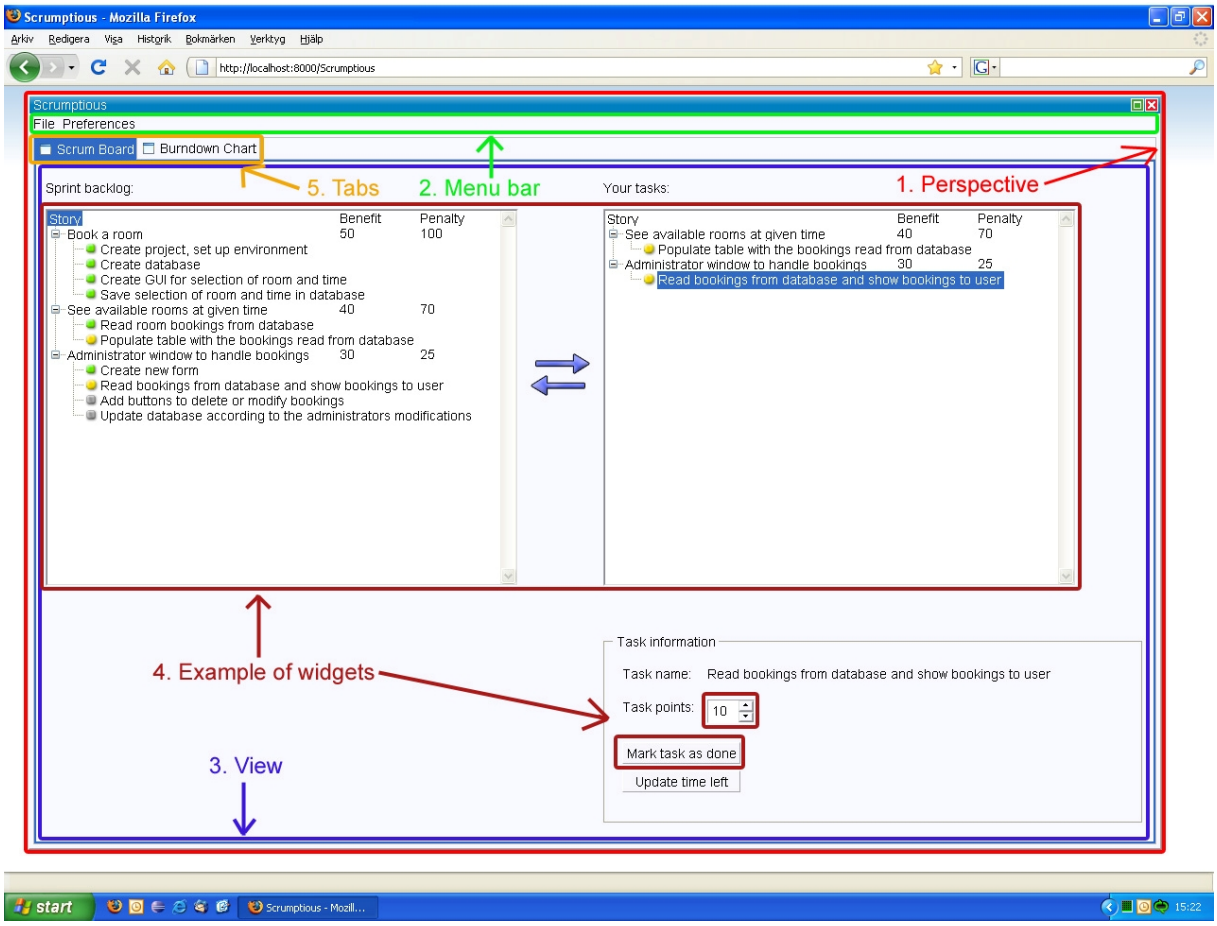

Figure 2-13 Screenshot of a RAP application

2.6.2.1 Manifest

When compiling a RAP application, the Java compiler which is a part of Eclipse will create a bundle [section 2.5.3] which is an application that can be run on the Equinox operating system [section 2.5.4]. To keep track of the resources in this bundle, a file called Manifest.MF is included. This file contains information about the bundle, such as the name of the bundle and the location of the activator class [section 2.6.2.5].

2.6.2.2 View

The view classes are the classes where all user interface elements are created and defined such as buttons, list boxes and text boxes. In an application, several views can be visible at once. In Figure 2-13, an example of a view can be seen. The view is inside the blue border, and contains widgets.

2.6.2.3 Perspective

When launching a RAP application, one or more perspectives must be defined. A perspective is a class that defines which views [section 2.6.2.2] will be visible in the window. It also defines where in the window the views will be located and the size of the views. The RAP application must have a default perspective, but the perspective can be changed during runtime of the application.

The perspective can display many views at once, either by showing them in tabs (one view per tab) or by showing them in the same tab, by giving them a limited area each. [26]. In Figure 2-13, an example of a perspective can be seen. The perspective is inside the red border, and contains, in this case, a menu and two views which are separated into two tabs.

2.6.2.4 Application

This is the main class of a RAP implementation and provides the name of the application. This class contains a run method which defines what to do when the application is launched. [26]

2.6.2.5 Activator

The activator class controls the life cycle of the application. This class defines what will be done when the plug-in starts and stops. [29]

2.6.2.6 Advisors

A RAP application contains three advisor classes which control the workbench and provide the foundation for the application.

1. **ApplicationWorkbenchAdvisor**: This class provides the name of the default perspective [section 2.6.2.3] to the workbench. [26]

2. **ApplicationWorkbenchWindowAdvisor**: This class controls the window that the RAP application creates. The class lets the developer define the title, window size and which window components that will be visible, for example if the window will have a toolbar or not. [26]
3. **ApplicationActionBarAdvisor**: The ApplicationActionBarAdvisor class controls the window bar, for example the status bar, tool bar and menu bars. The class defines which items that will be in the window bar, and what actions or events that will be triggered upon user interaction with the items. [26]

3 The Scrum prototype

3.1 Introduction

This chapter will explain the three main building blocks in the creation of the Scrum prototype, namely:

1. Drag and Drop [30] widget
2. Database design
3. Scrum prototype system design

These blocks are all needed to make the Scrum prototype work properly.

The drag and drop widget was the number one priority from Tieto, without it they would probably not consider using the prototype.

The widget itself is created in a way that it can be reused in other projects with only minor changes needed. The drag and drop widget had to be created since RAP does not support the drag and drop feature by default yet.

A database is needed since it is obvious that the data has to be stored somewhere. Storing data in a database is a relatively simple and a powerful way of storing data (i.e. easy to read/write and make backups).

In the Scrum prototype system design part the focus will be on design choices since they are easy to understand and give a good overview of the prototype. To give some more detailed examples, there will be some code fragments in this section.

3.2 Drag and Drop

3.2.1 Background

Since the RAP technology is a relatively new technology, it does not support the drag and drop feature [30] yet. However, the developer of a RAP application can create a custom widget which supports drag and drop within this custom widget. This is done by implementing a Java class which holds the server side data for the widget and a Qooxdoo JavaScript file which is a file implemented using Qooxdoo. The server part and client part of the custom widget are connected using an adaptor.

The widget will be used for two purposes:

1. Assigning users to a product
2. Planning/maintaining the different backlogs

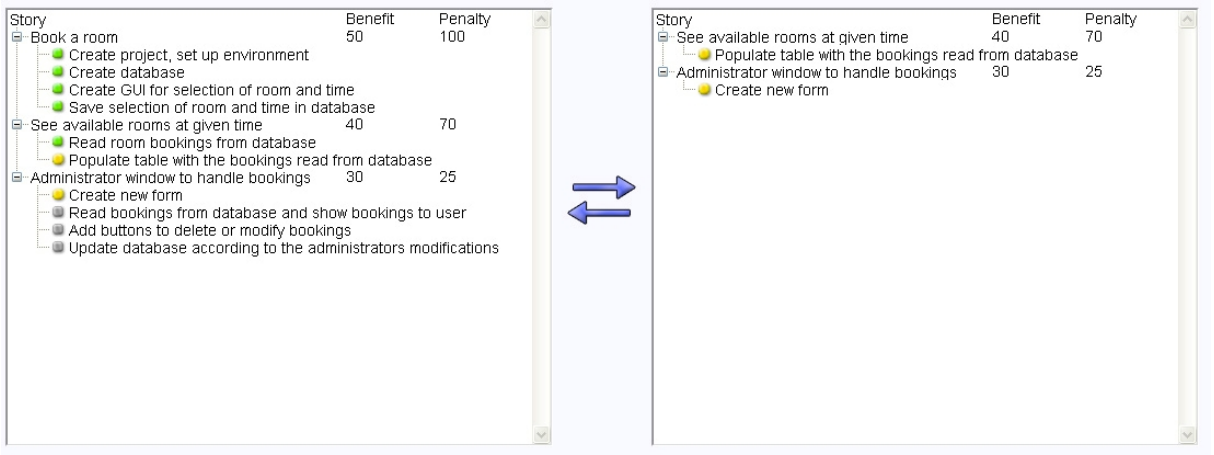

Figure 3-1 The drag and drop widget

Figure 3-1 shows the custom drag and drop widget when populated with the stories and task from sprint one in the Scrum example in section 2.2.1. The user can select a task in the left tree and drag it over to the right tree. The icons to the left of the task nodes indicate the status of the task, where a green icon means that the task is done, a yellow icon means that the task is checked out and a gray icon mean that the task is not checked out yet.

The widget is composed of two tree structures, one on each side in the picture above. The tree structure is very similar to the one used in the Explorer in Microsoft Windows XP under the folder hierarchy.

The reason for having a drag and drop feature was that Tieto felt that no matter how rich featured the rest of the Scrum prototype was, it would probably never be used since the drag and drop feature is such a natural part of the Scrum planning process. The feature makes it much easier for the users to handle a sprint planning session and to commit to tasks simply by dragging the stories or tasks from one list over to another. If the drag and drop widget had not been implemented, the users would have had to commit to stories either by using a context menu, or by pressing a button after they had selected the right task. Both alternatives require more interaction from the user than using drag and drop, which is a disadvantage especially when the user commits to many tasks.

The drag and drop widget is able to handle the following:

- Populate two tree widgets with data
- Handle drag and drop between these two trees
- Read data from both the trees

3.2.2 Creating the widget

To create a custom widget, a number of predefined classes must exist to enable the widget to function. In Lange [29], there is a basic description of the steps that have to be done when creating a custom widget. The basic steps for creating a custom widget are:

- Creating the Java part of the widget (server side of the widget)
- Creating the JavaScript and Qooxdoo part of the widget (client side of the widget)
- Creating the connection between Qooxdoo and Java

In the subsections below, these steps will be explained in more detail.

3.2.2.1 Java

The Java code is the server side of the widget. This part's main task is to hold data and therefore consists of a number of set and get methods. The Java widget holds both data that will be passed to the client part of the widget and data that is read from the client part. For example, when creating the Drag and drop widget, a number of elements is added to a list in the Java part of the widget. These elements are passed to the client part of the widget, which populates the tree structures with these elements.

Whenever the Java part of the widget changes, e.g. a new element is added to the widget, the renderChanges function is called in the LCA, and the elements are sent to the client part, and are added to the tree.

3.2.2.2 JavaScript/Qooxdoo

This is the client side of the drag and drop widget. This part of the drag and drop widget can be divided into three smaller parts each of which provides necessary functionality for this widget to work properly. The three parts are:

1. Defining the widget
2. Populate the widget with data
3. Read data from the widget

Defining the widget

This part handles the visual look and functionality of the widget. The functionality of this widget is how it will behave when a drag and drop is performed.

The widget is built on two predefined Qooxdoo objects called trees (the location of this tree object in the Qooxdoo framework is qx.ui.tree.Tree).

Three event listeners have to be added to each of the two tree objects so the following events can be handled:

1. Drag over
2. Drag out
3. Drag drop

The event drag over occurs when a user drags an object from one of the trees and holds it over the other tree. This is to help the user to understand where he/she is actually trying to drop the tree node object. This is shown by changing the font style of the tree folder object (e.g. user name) currently marked by the mouse pointer, to also include an underscore.

Similar to the drag over event a drag out event has been implemented. This event is trigged when a user moves a selected object away from the tree folder. This is to help the user understand that a drag and drop cannot be performed. When the event is triggered the underline which was shown by the drag over event is removed.

Finally there is the handle drag drop event which is triggered when a user drops the dragged object onto the other tree. This event removes the selected object from the source tree and adds it to the tree where the user drops it.

Populate the widget with data

Data are sent from the LCA adaptor to the Qooxdoo code as an array of strings. The first element in the array is a new tree folder and the rest of the elements in the array are children to that folder.

Pseudo code example:
Array[0] = "story1"
Array[1] = "task1"
Array[2] = "task2"

This is repeated for every story/user.

Read data from the widget

Data are sent from the Qooxdoo part to the LCA adaptor in an XML [50] format as well as plain text format.

Example 1 dragging users:

```
<data>
        <node>
                <name>Username</name>
                <subnode>E-mail</subnode>
        </node>
        <node>
                <name>Username2</name>
                <subnode>E-mai2l</subnode>
        </node>

</data>
```

Example 2 dragging task at the Scrum board:

These three parameters are sent every time a new drop occurs:

1. *Destination*
2. *Parent*
3. *Child*

Destination is which of the two trees where the element is dropped.

Parent is the parent element e.g. story.

Child is the child element e.g. task.

The reason for using two different types for sending data back to the LCA is that the first one is triggered only when a button is pressed down to confirm the drag process.

The second example at the Scrum board is triggered at every drop, this is because the confirm button was causing too much buttons to click and confused users.

The XML string is then parsed and matched against a sprint for the example above. That is since the Qooxdoo part does not have a unique identifier. This gives limitations on that every sprint cannot have two or more stories with the same name.

3.2.2.3 LCA

LCA [29] is an abbreviation for Lifecycle Adaptor, and is an adaptor between the JavaScript code and the Java code. This adaptor has some inherited methods that are called automatically at specific times in the widget's life cycle. The following list shows the inherited methods and a short description of them:

- renderInitialization
  This is the method that gets called at the initialization of the widget. This function often contains a call to the JavaScript codes constructor. [29]

- renderChanges
  When the Java part of the widget changes this method gets called. The developer can use this method to pass data from the Java part to the JavaScript part of the widget. This is done by calling a function in the Java part, and passing the return value to a function in the JavaScript code. [29]

- renderDispose
  This method gets called when the widget is disposed, and can be used to call methods on the JavaScript side if anything has to be taken care of before destroying the widget. [29]

- readData
  This method is basically the opposite of the renderChanges method. When the user interacts with the widget at the client side (e.g. when the user drags an item from one tree to the other) this method gets called. ReadData can then be used to read any property from the JavaScript and store in the Java class. [29]

- preserveValues
  This method is used to temporarily store data if the widget is disposed and then recreated. When reinitializing the widget, it can be reinitialized in the same state as when it was disposed. [29]

With the help of these methods, communication between client and server is possible. The developer can use the methods to read and write data to both sides of the widget.

3.3 Implementation

This section contains information of the implementation of the Scrum prototype. The database design (i.e. tables and its attributes in the database) and system design (i.e. basic classes and their connections) are presented in the subchapters.

3.3.1 Database design

In this section the design of the database will be presented. Figure 3-2 shows an overview diagram of the tables in the database. There is a more detailed description of the tables in the database in appendix A.

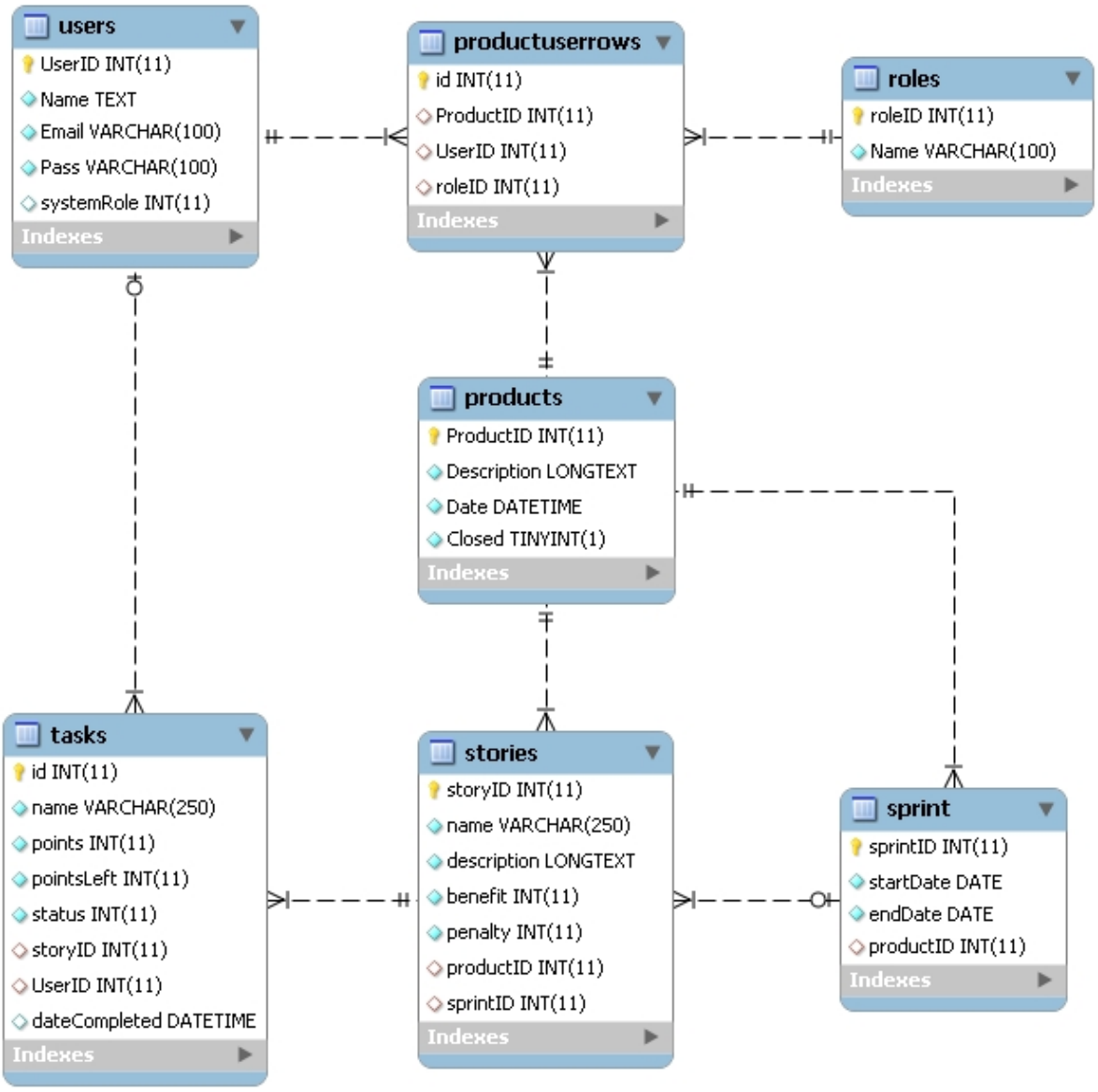

Figure 3-2 Diagram of the database

The tables products, tasks, sprint, users, stories and roles are used to store data and information of the corresponding container classes [section 3.3.2.1].

Between the tables products and users there is a many-to-many relationship [46], which means that a junction table is needed. The table productuserrows is the junction table in this case. The table consists of four columns: id, productID, userID and roleID. The id is the primary key of the table and is therefore set to auto increment. The productID is a foreign key to the productID column in the products table, the userID is a foreign key to the userID column in the users table and the roleID is a foreign key to the roleID column in the roles

table. This basically means that a row in the productuserrows table connects a user to a specific product and defines what type of role the user has in the product.

3.3.2 Scrum Prototype System design

This section will cover the system design of the Scrum prototype, the relationships of the components and what part they play in the Scrum prototype. In the picture below, an overview of the system design is presented.

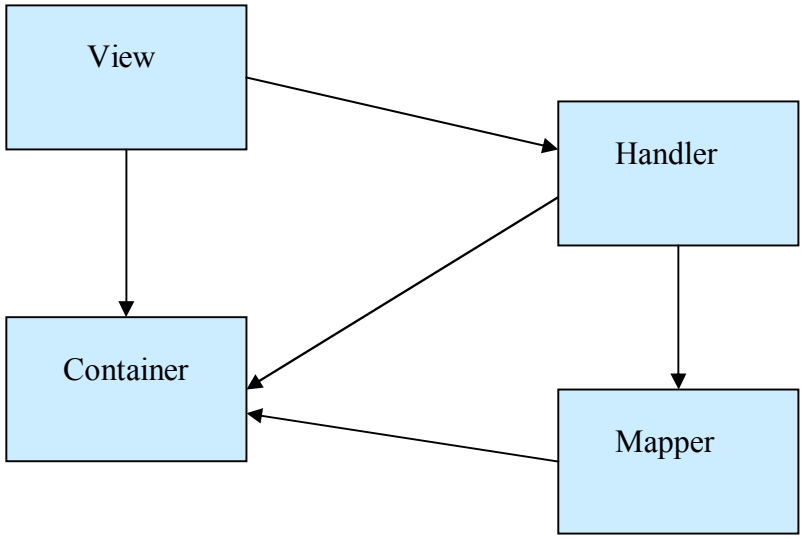

Figure 3-3 Overview of the classes in the RAP application

The containers are the objects which the Scrum prototype uses (e.g. Products, Users etc.) and the handlers are the singleton objects which operate at the container objects. The view is the user interface and is what is displayed on the screen and the mapper is the link between the database and the rest of the system to make it easy to change data source. The arrows in the figure represent the relationships between the classes. For example, views have knowledge and access to containers and handlers, but not vice versa. A more detailed explanation of these components is available in the chapters below.

3.3.2.1 Containers

In the Scrum prototype, seven container classes have been implemented:

1. Product
2. Sprint
3. Story
4. Task
5. User
6. Role

These classes are used to store information read from the database during a session of the Scrum prototype. These classes are connected to each other i.e. some of the classes contain instances of other container classes. An overview of this can be seen in Figure 3-4.

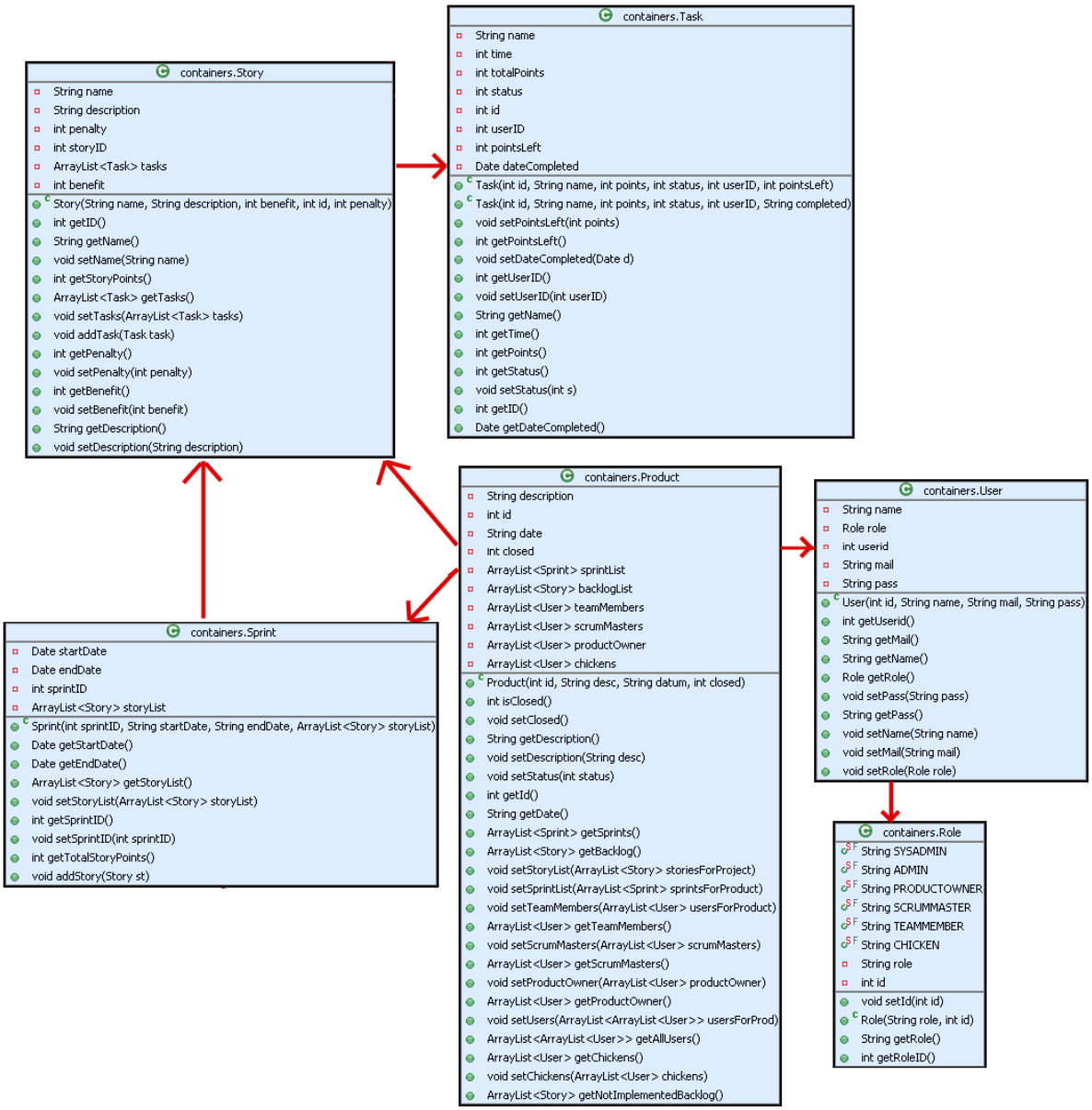

Figure 3-4 UML diagram of the container classes

Below is a brief explanation of the content of each of the container classes.

The **product** container is a class that contains information about a product, including a description of the product, the date when the project for creating the product was started, a list of all sprints, a list of all stories and tasks in the product backlog and a list of all users connected to the product.

The **sprint** class contains information about the sprints, such as the start date and end date of the sprint, and a list with all stories connected to the sprint.

The **story** class contains information about a story, which is the name of the story, the penalty of not implementing the story, the benefit of implementing the story and a list of tasks connected to the story.

The **task** class contains information about a task, such as a description of the task, and a list of time log items to receive information of how much time a user spent on the task.

The **user** class holds information about a user, including the user's name, what role the user has in the project and the users e-mail address.

The **role** class is a container which holds a string indicating a specific role. There are six different roles in the system:

The **time log** class contains information of how much time a user has spent on a specific day. This container is connected to a task, and will give an overview of how much time spent on a task.

3.3.2.2 Handlers

The code of the Scrum prototype contains different handler classes. A handler is a class that operates on container classes. This is a design decision to enforce the single responsibility principle [36] this design pattern makes the system more discrete.

All handler classes use the singleton pattern [37] which means that there can only be one instance of the handler objects. This instance is therefore shared by all the users' of the Scrum prototype.

ProductHandler

The ProductHandler is the most frequently used handler in the Scrum prototype. The handler is used to handle most of the information regarding a product, such as reading stories, sprints and users for a product and handle the status of the product (i.e. open or closed).

The product handler is using a modified identity map [38]. This is to make sure that the product handler does not cache too many product objects in memory.

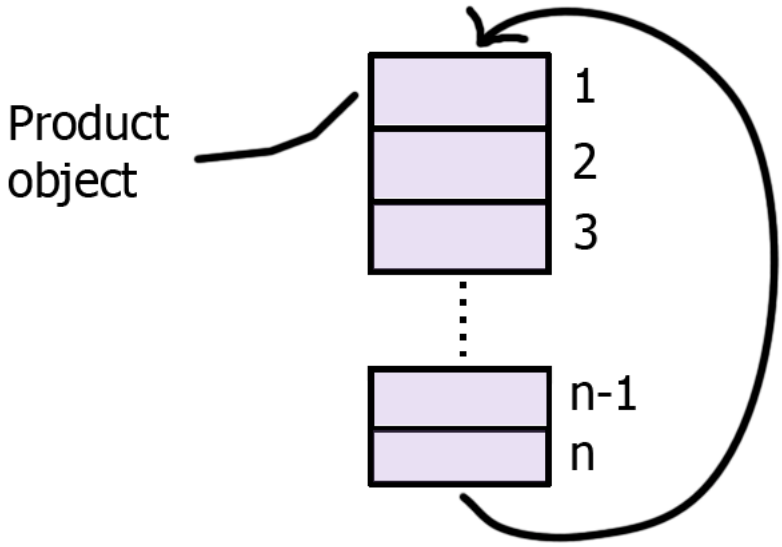

Figure 3-5 Illustration of the identity map

The number n can be altered to ensure the best performance possible depending on how many resources the server has.

When the identity map is filled up and a new product is read into memory, the server puts the new product at position 1 and at the same time removes the previously stored object from position 1.

UserHandler

The UserHandler class is used to handle all operations affecting the User objects. The class also keeps track of which users that are logged in. This is done by storing the User object at the client side, which is a way of making each client browser unique. So the server does not mix up which user it is.

Some of its methods are:

- getAllUsers
- setCurrentUser
- getCurrentUser

The setCurrentUser and getCurrentUser may seem unnecessary since they only sets and gets the active user at the client. However, the benefit of having these functions is to make it easier to convert the RAP application to a stand-alone RCP application.

3.3.2.3 Database mapper

In the Scrum prototype there is a mapper which role is to handle the read and write of data from the database. All the information regarding the database is stored in these two classes.

1. ReadFromDB
2. WriteToDB

This is to make it easier to migrate the Scrum prototype to another data source. Another feature to facilitate a change of data source is that both the read and the write class operate on container objects instead of using database logic (e.g. indexes in the database).

An example from ReadFromDB

The method getStoriesForProject takes a product object as parameter and creates an empty ArrayList [41] which can store story objects. Then the method queries the database for all the stories that belong to the specific product. For each of the stories returned from the database, a new story object is created and added to the ArrayList. If everything went well the ArrayList is returned otherwise a printout with an error message is displayed.

```
public ArrayList<Story> getStoriesForProject(Product product)
{
    ArrayList<Story> list = new ArrayList<Story>();
    ch.openConnection();
                try
                {
                    ps = ch.getConn().prepareStatement("SELECT name, penalty, stories.storyID, benefit,
                            stories.description from stories INNER JOIN products ON stories.ProductID =
                            products.ProductID WHERE products.ProductID = " + product.getId());
                    ResultSet rs = ps.executeQuery();

                    while(rs.next())
                    {
                        Story story = new Story(rs.getString(1), rs.getString(5), rs.getInt(2), rs.getInt(3),
                                rs.getInt(4));
                        list.add(story);
```

```
                story.setTasks(getTasks(story));
            }
            ps.close();
        }
        catch(Exception e)
        {
            System.out.println("READ STORY: " + e.getMessage());
        }
        finally
        {
            ch.closeConnection();
        }
        return list;
}
```

Some examples from WriteToDB

To prevent redundancy in the WriteToDB class, there is a private method which handles the queries to the database, called “Write”. Then there are a number of public methods that creates the specific SQL statements which are sent to the Write method.

```
private void Write(String sql)
{
    ch.openConnection();
    try
    {
        ps = ch.getConn().prepareStatement(sql);
        ps.executeUpdate();
        ps.close();
    }
    catch(SQLException ex)
    {
        ex.printStackTrace();
    }
    finally
    {
        ch.closeConnection();
    }
}
```

The writeSprint method takes a product object and a sprint object as parameters. The product object is the product to where the new sprint will be added to. The SQL statement is created as a string and then passed to the Write method which executes the statement. Note

that the writeSprint method only writes to an empty sprint. There is another method to write the stories included in the sprint.

```
public void writeSprint(Product prod, Sprint sp) {

    String sql = "INSERT INTO sprint (startDate, endDate, productID) VALUES('"+ sp.getStartDate() + "', '" +
    sp.getEndDate() + "', " + prod.getId() + ");";

Write(sql);
}
```

3.3.2.4 Views

The Scrum prototype consists of multiple views some of which are:

- CreateNewProduct
- PlanNewSprint
- ProductBacklog

The role of a view in this Scrum prototype is to display data to the user (e.g. listing all the users). A view consists of one or more widgets, which are able to both display data as well as handle the input of data.

The input of data in this section refers to a user typing or performing some sort of interaction with a widget (e.g. drag and drop). For example when the user creates a new product, the name of the product has to be entered somewhere and the view CreateNewProduct handles this.

Figure 3-6 shows the view used to create new products. The view contains a wizard with three steps, one for entering the name of the product, one for selecting the Scrum Master and one for selecting a product owner.

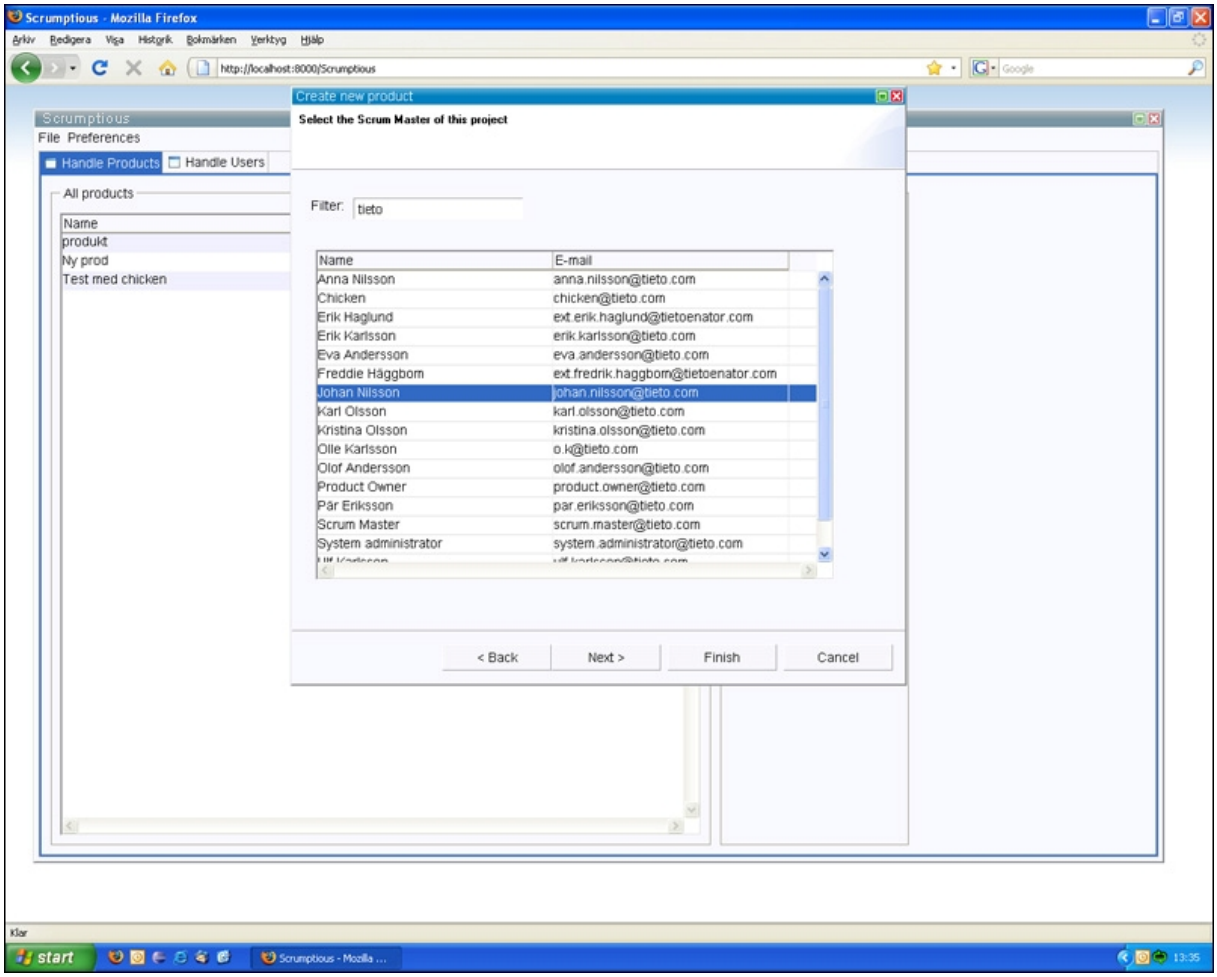

Figure 3-6 The view used to create a new product

3.3.2.5 Example of write flow

This is an example of creating a new product in this Scrum prototype.

The user types in the name of the product and selects which team members that will be working on this new product. All this is done in the view CreateNewProduct.

The view uses these values to create a new instance of a product container object. That specific product object is passed to the product handler instance. The handler passes the product object to the mapper, which writes the product to the database. The handler then fetches the database primary key (productID) for this product. This id is used by the mapper to bind the selected users' to the product.

An illustration of this example can be seen in Figure 3-7.

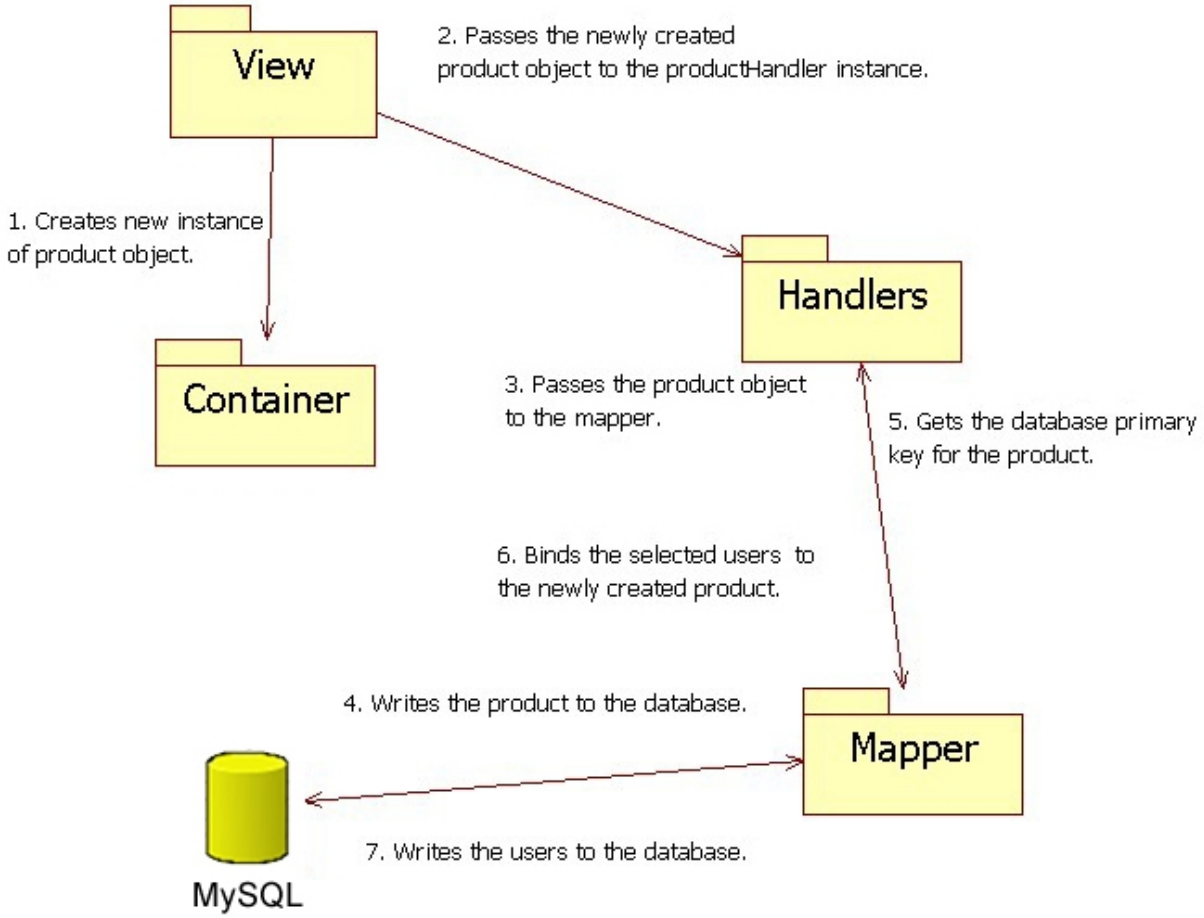

Figure 3-7 Steps taken when a new product is created

3.3.2.6 Example of read flow

This is an example of how a user requesting a product and shows the different steps the Scrum prototype can take.

A user requests a product; this could be done in the ProductBacklog view. The ProductHandler checks if the product exists in the identity map, if so the product is returned.

Otherwise the request is passed on to the ReadFromDB class in the Mapper package. The data is read and a new Product object is created. That object is passed back to the ProductHandler which then passes it to the current view where it is displayed.

An illustration of this example can be seen in Figure 3-8.

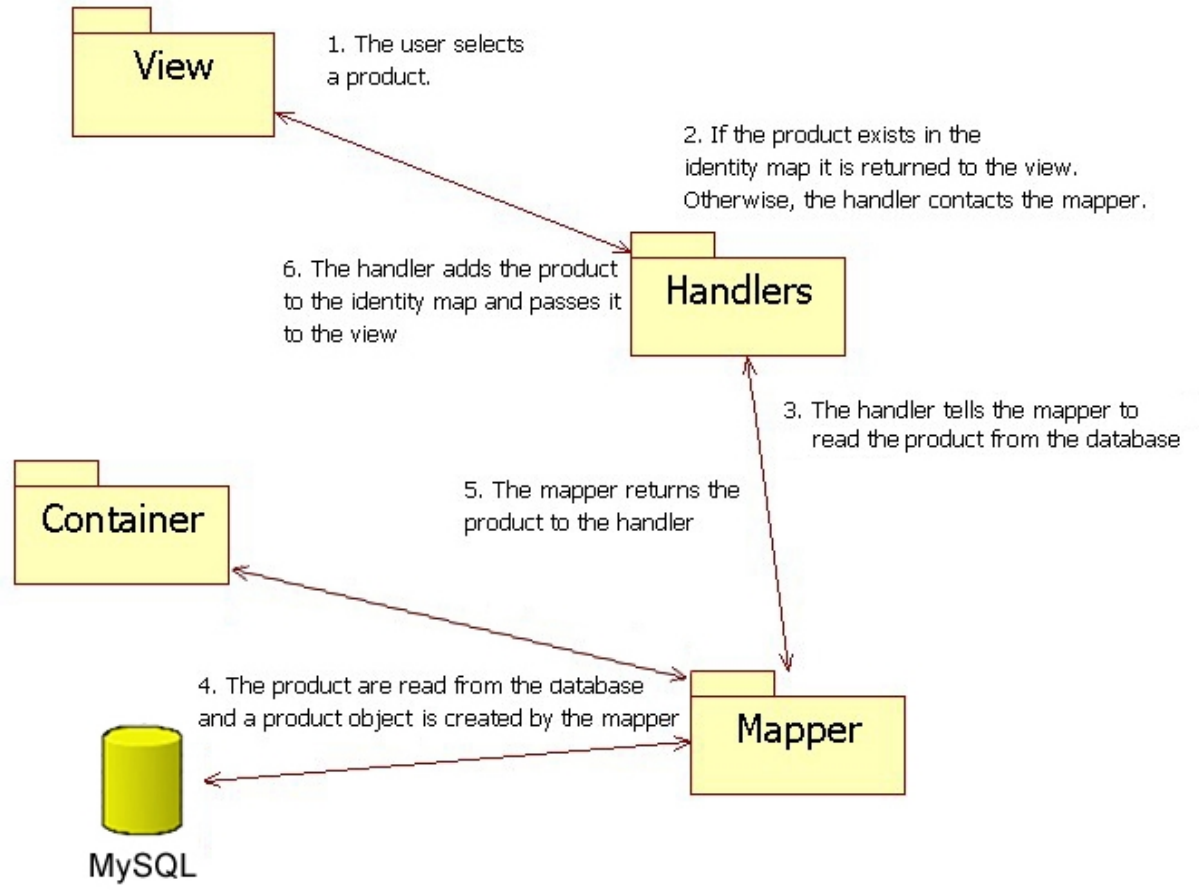

Figure 3-8 Steps taken when a product is read from database.

3.4 Summary

The Drag and Drop widget was created using three components: Java, JavaScript (Qooxdoo) and an adaptor between them. It is mainly used at the virtual Scrum board, when committing to a task. The database consists of the tables' products, sprints, stories, tasks, users, roles and productuserrows.

Regarding system design, the prototype consists of four components: view, handlers, containers and mappers.

4 The RAP evaluation and comparison

This chapter will contain an evaluation of the RAP technology used to create the Scrum prototype. A comparison between RAP and GWT (Google web toolkit) will also be presented, as well as a short description of the GWT.

4.1 GWT – an introduction

Both RAP and GWT provide the ability for developers to create Rich Internet Applications by developing applications in the Java programming language. The developer can use any Java editor to write the code, and GWT will then cross-compile [47] the Java code into JavaScript which can be launched on any major web browser. To do this, GWT has four major components:

1. **Java to JavaScript compiler**: This is the cross-compiler that translates the Java code into JavaScript code. The compiler is used when you want to test your application in a web browser.
2. **Hosted Web Browser**: When launching the web application in the hosted web browser, the code is not compiled into JavaScript, it runs as Java code in a Java Virtual Machine. The main advantage of this is that the developer can make changes in the Java code and see the result in the web application without recompiling.
3. **JRE Emulation Library**: This library contains JavaScript implementations of the main Java classes and functions.
4. **GWT Web UI class library**: This library contains interfaces and classes used to create the widgets in the GWT applications, for example buttons, list boxes and tables.

The Google Web Toolkit was released in 2006, and version 1.0 of GWT was released the same year. The most recent version is 1.6 [52]. GWT is open source.

4.2 RAP vs. GWT

In this section, a comparison between the RAP and GWT technologies is presented in the form of a discussion of the similarities and differences between the technologies. For an optimal comparison, an application would have to be developed in RAP, and then the same application would have to be developed using GWT. These applications could then be used to perform several tests in order to compare the two implementations. Due to time limits and the fact that this was not required in Tieto's original specification, a GWT application was not developed which means that no were tests performed in this project. However Russel and Rubel [51] have compared the performance of a RAP application and the same application developed with GWT. This study will be used in this discussion as an indicator of performance between the two systems.

4.2.1 Similarities

Both technologies serve the same main purpose; they allow the developers to create Rich Internet Applications consisting of JavaScript code by developing the application using the Java programming language. Both technologies also support styling of the application by using cascading style sheets (CSS).

Since both technologies are based on the Java programming language, both GWT and RAP can be tested using JUnit [48] and during development, the code can be debugged as when developing a regular Java application.

4.2.2 Differences

RAP and GWT serve the same purpose, but in different ways. In this section, the differences between RAP and GWT will be presented and discussed.

4.2.2.1 Communication

RAP puts most of the processing work on the server side, which means that the client part only handles the graphical user interface and the events triggered by the user. The events are

sent to the server where they are handled and processed. After processing, a response is sent back to the client and the graphical user interface updates its content if required. A diagram showing the communication steps in a RAP application is presented in Figure 4-1.

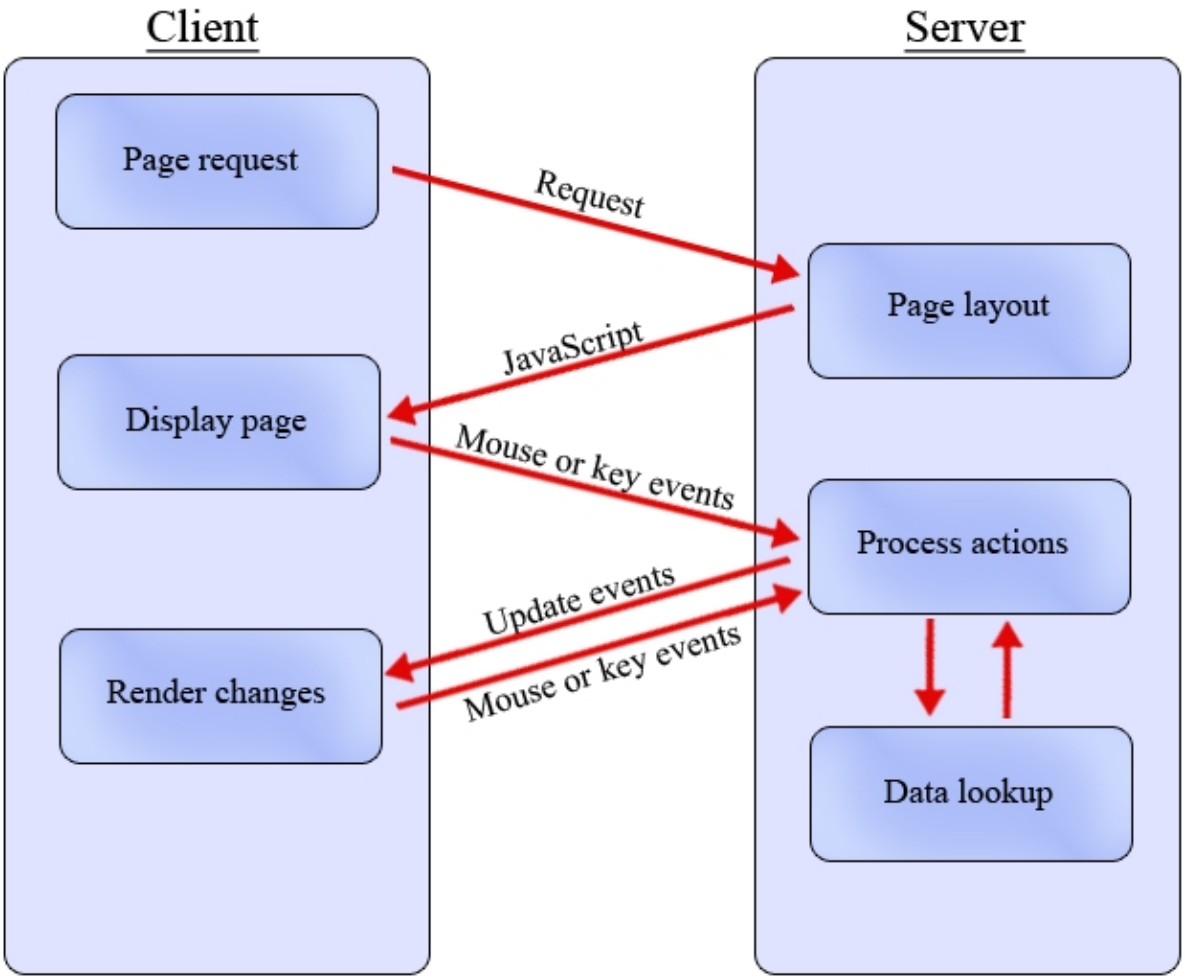

Figure 4-1 Communication flow in a RAP application

The GWT technology handles the communication in a different way. This technique does not process the events on the server, they are instead processed at the client. A diagram showing the communication steps in a GWT application is presented in Figure 4-2.

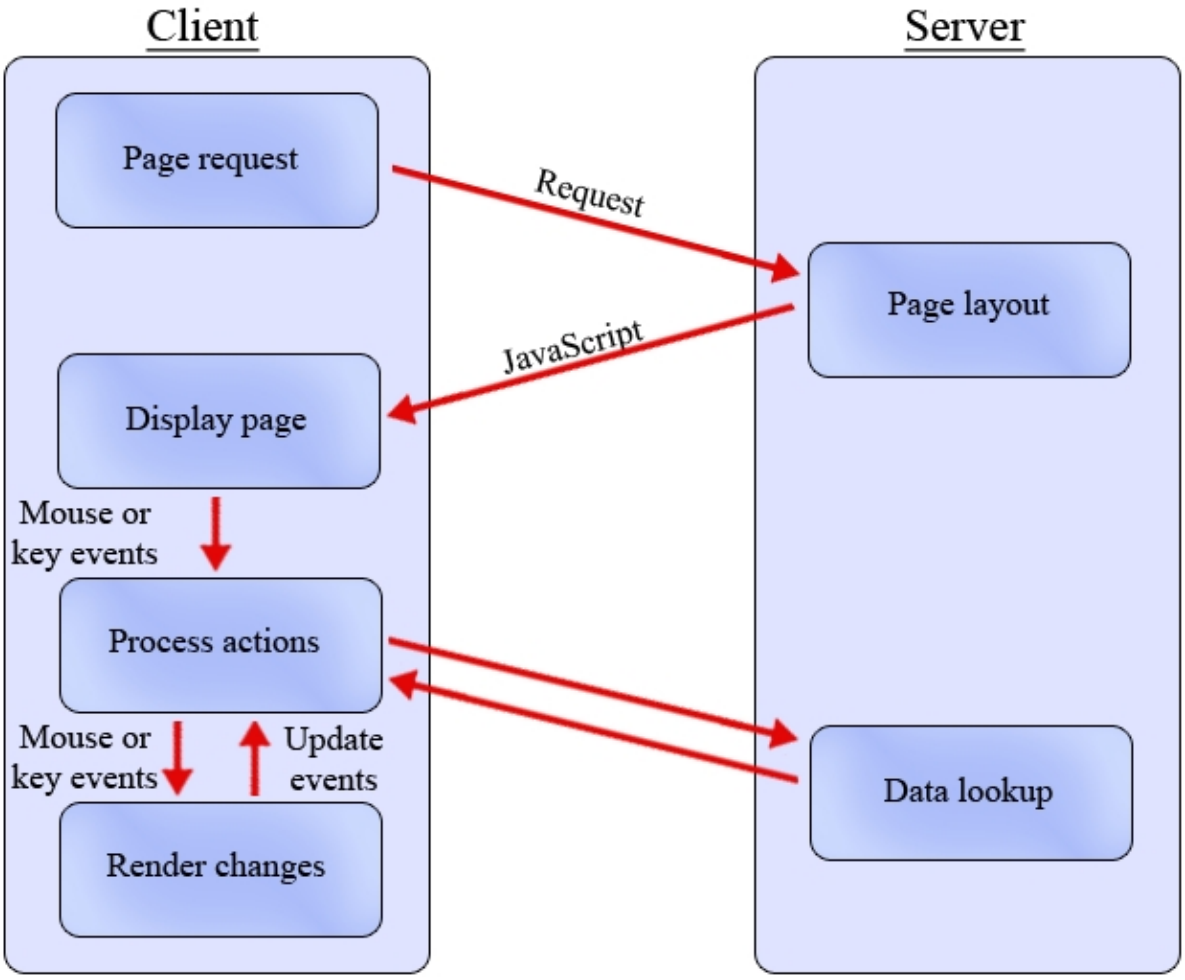

Figure 4-2 Communication flow in a GWT application

In RAP, the communication is set up automatically which means no extra work is needed by the developer. In GWT the communication is set up by the developer, which needs more work, but gives the advantage of more flexibility, such as the option of caching information at the client side and the option to pre-fetch data as a background operation.

One advantage of handling all events at the server side as RAP does is that the clients need a minimum of resources. Even if the processing of events often does not require a high level of resources, it can be advantageous to perform all processing on a server instead of on the clients.

The disadvantage of this method is that greater communication between client and server is required.

For GWT the opposite is the case, the advantage is that less communication is required, but the disadvantage is that the processing is performed at the client side.

4.2.2.2 Compiling

GWT

The GWT example application [53] takes 18.175 seconds to compile. The reason as to why the GWT compiler is slow depends on two factors. The first is the fact that the compiler compiles at least four different versions of the application - one for each major web browser (Internet Explorer, FireFox, Safari and Opera). If the user application is available in two or more languages, the compiler will produce one version for each language.

For example, if the application is available in four different languages the result will be 16 different versions of the application (4 languages * 4 browsers).

The benefit of this compilation method is that only the application for the user's current web browser and language selection will be run, saving memory and bandwidth. The compiler can also tweak each application for every web browser for known bugs without interfering with the other web browsers.

RAP

The compiler used by the RAP reads RCP code as input and produces an executable application using either the RCP or the RAP technology. This is since the RAP and RCP code is identical and compilation is done against the same set of Workbench, JFace and SWT APIs.

If the compiler is to produce a RAP application, the compiler simply substitutes the regular used libraries (in Workbench, JFace and SWT) against web enabled versions of them (e.g. Simple Widget Toolkit => RAP Widget Toolkit).

Regarding compile time, the Scrum prototype which is more complex than the GWT example application, takes about 1 second.

4.2.2.3 Performance

The following performance tests were performed by Russel and Rubel [51]. They wrote a RAP and a GWT application with the same features, and measured the time taken for different operations in these applications. The results can be seen in Table 4-1.

		RAP		**GWT**		
	Connection		Cached		Cached	GWT advantage
Load page	A	8	5	**5**	**3**	**35%**
	B	3	3	**3**	**2**	**15%**
Load folder	A	3	n/a	**2**	**1**	**50%**
	B	3/4	n/a	**3/4**	**3/4**	**0%**
Sort List	A	2	n/a	n/a	**1**	**50%**
	B	½	n/a	n/a	**1/2**	**0%**

Table 4-1 Time comparison between RAP and GWT. The unit used is seconds

Command	**RAP**		**GWT**	
	Packets	Bytes	Packets	Bytes
Sort List	63	25390	**11**	**4661**
Load folder	**8**	**2900**	8	3827
Load folder again	6	2602	**4**	**349**

Table 4-2 Comparison of transferred data when performing different commands

In Table 4-1, connection A is an average broadband connection used at home and connection B is an average broadband connection available at a work site.

Table 4-1 shows that in the performance test, GWT is faster then RAP in four of the six tests, and that both technologies is equally fast in the two other tests. The reason for GWT's speed advantage will be discussed in the subsections below.

The reason as to why a value for the cached test of RAP is not applicable in some tests is that RAP does not cache all information at the client (see Figure 4-1). In the sort list test, the

opposite is true for GWT; the non-cached test cannot be performed because the list is cached at the client when the list is loaded. This implies that the list is always cached at the client when a sorting operation is performed on the list.

Load page

The reason as to why GWT is slightly faster than RAP when loading pages/applications is that GWT uses a technique to bundle the images in a GWT application. This means that the server combines several images into one large image before transmitting it to the client. This gives a gain in performance since it is more efficient to load one larger file than to load many small files. At present, this is a significant advantage for GWT, but since RAP is under development and has a similar technique scheduled for a future release, this will even up the performance between RAP and GWT.

As mentioned in section 4.2.2.1, RAP processes events at server side. This means that the client and server have to send data between each other every time the user triggers an event, which results in more data transmission compared to GWT.

Load folder

The application built and tested in Russel and Rubel's experiment [51] contained a folder hierarchy similar to the one which is used in the explorer in Microsoft Windows XP. When expanding one of the folders, the user could see the content of that specific folder.

In their experiment, the test shows that the first time a folder is opened; RAP transfers less data then GWT. This is since the RAP application only transfers the visible entries in a list to the client, not the entries that the user has to scroll to see. However, in the GWT version, all the content of the folder is transferred and cached at the client. See Figure 4-3 for an illustration of this difference. The red square in the figure represents how many elements each of the techniques loads when displaying the list. GWT lets the developer decide on his/her own if every callback should result in a new transfer of content which makes GWT more flexible.

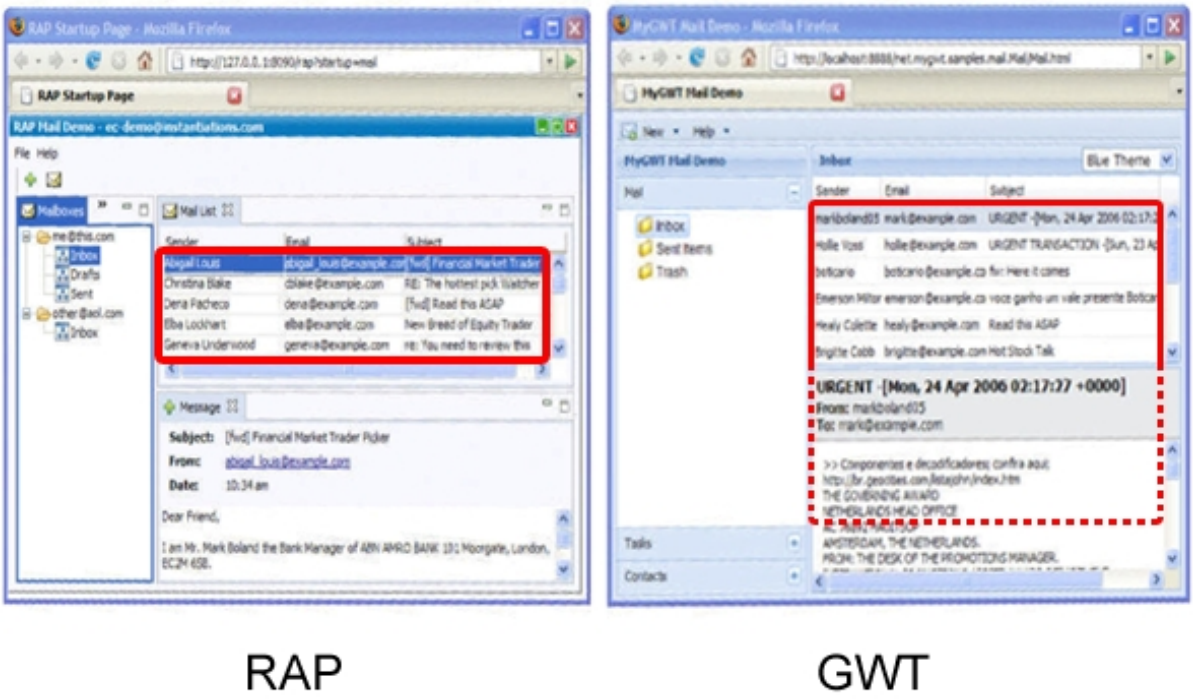

Figure 4-3 Difference between RAP and GWT when loading lists

Reload a folder

This test was made by loading a folder, then loading another folder and then loading the first folder again. The packet and byte values in the table are measured when loading the first folder again. In this test there is a significant difference between RAP and GWT. RAP sends 25% fewer packets and sends 10% less data compared to when the folder was loaded for the first time. For GWT the same numbers are 50% respectively 91%. The reason to this significant difference is that GWT caches data on the client whilst RAP keeps most of the data at server side. When the GWT application loads the folder for the first time, its content is cached on the client. When the same folder is loaded again, the content is located at the client, which means that less data has to be transferred from the server. When using RAP, the content of the folder is not cached on the clients, which means that the data has to be resent from the server when the folder is loaded again.

Sort list

This was performed by letting a user click six times on different attribute columns resulting in six different sorts. Since RAP has to send every callback to the server, the result is that more than five times as much data has to be sent compared to GWT.

GWT first caches the whole list at the client and performs every operation on the cached list. This is why the list only has to be transferred once from the server to the client. Again this is a choice made by the developer of the GWT application.

4.3 The RAP/GWT Comparison - Conclusion

In this section a summary and a conclusion of the comparison in section 4.2 will be presented. First, a listing of advantages and disadvantages will be shown for both RAP and GWT, followed by a short summary of the comparison of the technologies.

4.3.1 RAP

The following table shows the advantages and disadvantages of the RAP technology:

Advantages:	Disadvantages:
• Single sourcing: develop one application for the web and/or as a standalone application. • Easy to switch to RAP from RCP (in theory no difference) • Uses the same set of widgets for the web as for a standalone application • Open source	• All SWT widgets have not been implemented yet. • Relatively slow • Still under development

Table 4-3 Advantages and disadvantages of the RAP technology

The biggest benefit of using the RAP is the single sourcing feature. This means that one application can be developed and deployed as a RIA and/or as a standalone RCP. This is because the both platforms use the same set of widgets which are supplied via Java libraries. Applications already developed using RCP, can be reused and launched as a RAP application, and be run on the web.

Access to the whole Java library and Java API is possible.

The downside is that the RAP is still under development and all the SWT widgets and events have not been translated yet. The RAP is also slower since a lot of the work is done at the server side, which means that more data has to be transferred between the server and the client.

4.3.2 GWT

The following table shows the advantages and disadvantages of the GWT technology:

Advantages:	Disadvantages:
• Faster • Scale better • More efficient bandwidth use • Open source • One version for each major web browser	• No single sourcing • Long compile time • More complex to develop • Uses unique widgets for the web • Still under development

Table 4-4 Advantages and disadvantages of the GWT technology

GWT is faster and more efficient in its bandwidth use. It will also scale better since it does not allocate that many resources at the server side. The GWT compiler will compile a different version of the application for each major web browser, thus the compiler can tweak each version for better performance.

The downside is no single sourcing, which means that if a project is developed using GWT, it has to be deployed over the web. The long compile time can be frustrating during the

development. The developer has to learn a new set of widgets and how they work. However, this is also a problem for the RAP if the developer does not have any experience with the RCP.

Developing GWT applications is a bit more complex since for example all callbacks to the server have to be coded by hand, thus increasing the probability of error.

4.3.3 Summary

If you as a developer already are familiar with the RCP, a migration to the RAP will be easy, since it is based on the same concept and uses the same structure of code and classes.

If on the other hand scaling and speed is important, then GWT is the technology to use. However the RAP developers are working on speeding up the RAP technique, but it will probably not be as fast as the GWT due to its system design.

5 Conclusions and project evaluation

In this chapter, an evaluation of the project is presented as well as a section describing future work, i.e. the next steps to do in the Scrum prototype.

5.1 Evaluation of the technology

From this project, we have a positive impression of the RAP technology. The technique lets the developers create an advanced Web 2.0 application as well a standalone application without using any other programming language than Java. The biggest drawback of the RAP is speed. Compared to a standalone application, the RAP application is slower since it communicates more frequently with a remote server. This may however not be a problem if using the RAP where high speed network connections are available e.g. in a LAN [58]. The technology has some bugs and not all Java widgets and events are implemented, but the bugs and most of the Java API will be fixed in future releases. The main positive aspects of the RAP is the single sourcing feature e.g. the ability to develop an advanced Web 2.0 application in the same way as a stand alone application is developed.

If we had been asked to develop a web based application for a company, we would choose the RAP technology since it is good enough and has more positive aspect than negative. We recommend Tieto to use this technology since the developers already are familiar with the existing technology RCP. This will make the migration to RAP easy because the developers do not need to learn a new technology.

5.2 Evaluation of the project

Table 5-1 lists positive and negative aspects which arose during the course of the project.

Positive	Negative
• Implemented drag and drop • Produced a usable prototype • Supervisors – good knowledge • Environment – no major problems • Helpful newsgroup	• Prototype specification – prioritizing • Learning curve – steep • Lack of documentation

Table 5-1 Positive and negative aspects with this project

The overall impression of the outcome of this project is good. However the initial specification was too ambitious and the different features had not been prioritized at the outset. We have been working on our own most of the time, thus a lot of assumptions have been made of necessity from our point of view.

The learning curve was steep due to our lack of experience and the difficulty of finding documentation on the RAP and Qooxdoo. The latter point consumed a lot of time in the beginning of this project.

Overall this project has been more positive than negative. The environment and administration at Tieto worked very well. At the first day at Tieto, we were allocated two computers, an office and key cards. We also received computer accounts and e-mail addresses that we could use to contact our supervisors. The supervisors were easy to contact, either by mail or by an instant messaging system called Microsoft Office Communicator 2007 [57]. The supervisors had good knowledge in the areas of this project, which eased the work for us.

The RAP has a newsgroup which, due to lack of documentation, has been very helpful during the development. After asking a question in the newsgroup, a reply was generally given within four hours. The users in the newsgroup include the RAP developing team which means that almost any question regarding the RAP was answered.

All these positive aspects helped us create a useable prototype with a working drag and drop widget.

Overall, this project was educational in many aspects. It was interesting to learn about the RAP, which has a great potential for the new generation of web applications.

5.2.1 Time management

The time available for this project was 20 weeks at 50% workload, which is equivalent to 10 weeks at 100% workload. The project was divided into two parts: writing the documentation and developing the Scrum process handling prototype. The distribution of time between these two tasks was about 40% for the documentation and 60% for the Scrum process handling prototype, which means that 4 weeks were used to write the documentation and 6 weeks for developing the prototype.

The prototype development phase can be divided into four subparts:

1. Set up the environment
2. Learn the basics of RAP and Eclipse
3. Develop the Scrum process handling prototype
4. Create the custom DND widget

Although a time log was not kept during the project, an estimation of the time spent on each of these four tasks is 10% for setting up the environment, 10% for learning the basics of RAP and Eclipse, 50% for developing the Scrum process handling prototype (excluding all work with the drag and drop widget) and 30% on all the work with the development of the drag and drop widget.

In Figure 5-1 there is a pie chart showing the distribution of time in the project. The time unit used in the chart is days, where one day is defined as eight hours of work.

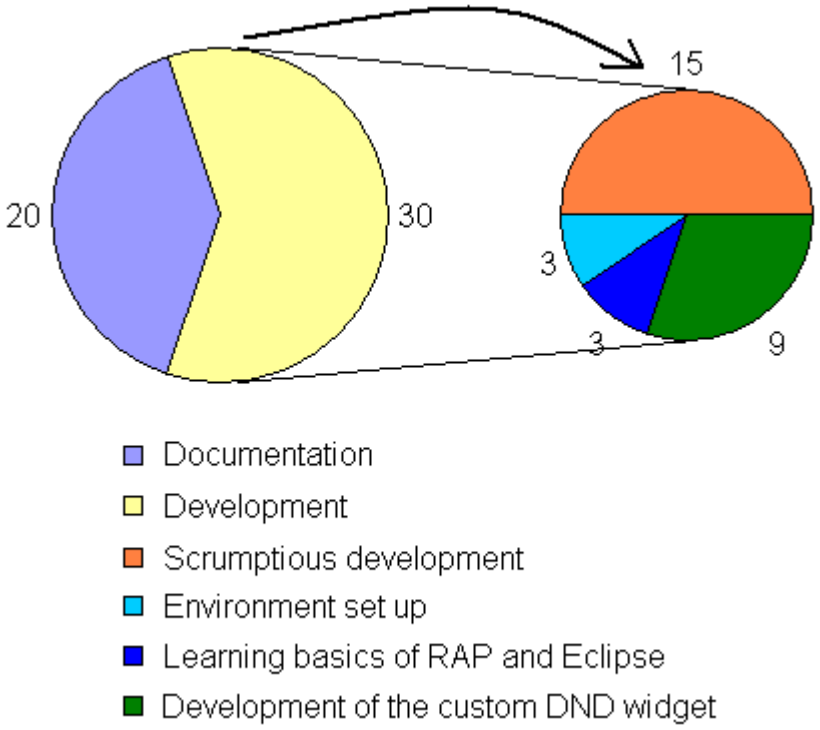

Figure 5-1 Distribution of time during the project

It has only been during the last month that the development process has gone smoothly, this is due to the long learning curve. Even though we did start developing at an early stage, it took long time to make any significant progress since the technique was new to us. In the beginning of the development we used trial and error to test code that we found in documentation and demo applications.

To reduce the time spent trying and testing code in this project, we could have performed more research and testing before starting this project.

5.3 Future work

Due to time constraints, all features of the specification have not been implemented. In this section, a list of these features and other future work in this project will be presented.

5.3.1 Unimplemented features

In this project, two features from the specification have not been implemented:

1. Support to add a backlog item or task in Scrumptious to JIRA
2. Create a link between Eclipse Mylyn and Scrumptious

JIRA [56] is a tracking system commonly used for bug and issue tracking. Connecting the Scrum prototype to JIRA would be useful. If for example a story has been implemented in one sprint and a bug has been discovered in the newly implemented story. A feature in the Scrum prototype to create a new issue in the JIRA system for tracking that bug would ease the work.

Eclipse Mylyn [54] is a task management system where the user can integrate tasks from for example Bugzilla [55] or JIRA. Once a task is integrated in Mylyn, the task can be monitored for work progress or updates. By using Mylyn, the user can gather specific tasks from different systems at one place.

5.3.2 The comparison

To obtain a better comparison, a small application should be developed using the RAP technology and then using the GWT technology. The reason to why no GWT application was developed in this project was that Tieto was more interested in the RAP technology and whether it is a useable technique. If a GWT application had been developed, there would have been too little time developing using RAP and the focus of this dissertation would not have been what Tieto required since the comparison part would have been the main focus.

However, future work would still include the development of at least a small application using GWT in order to be able to perform tests on this application, such as performance tests

and time measurements. Even if Tieto decides to work with RAP, the testing of the competitor GWT is recommended before making a definite choice.

5.4 What we have learnt

We have learnt to develop a Web 2.0 application. It has been both challenging and fun. Challenging in the sense that neither of us had any experience with the RCP and the Eclipse IDE before starting this project.

Eclipse is easy to learn at least if you have experience from other IDE's (e.g. Visual Studio) which we both had. The RAP was more challenging to learn, but after some analysis of demo applications showing the functionality and some help from the RAP newsgroup we managed to learn the basics of RAP. Since most of the programming is done in Java, it was pretty straight forward.

The creation of the DND widget was the part which gave us the most problems. We did not have much experience with JavaScript and no previous experience from Qooxdoo. The learning curve was steep and required a non-trivial amount of time. The result however is a custom widget that supports drag and drop, something that at the beginning of this project seemed too time-consuming to include.

It has been interesting to have worked in an industry environment and to have experienced how projects are handled and managed in the industry. We hope Tieto is happy with the result of the prototype as well as our evaluation regarding development using the RAP.

References

[1] http://www.eclipse.org/org/ 090217

[2] http://www.cs.kau.se/cs/education/courses/dvgc02/08p4/ohslides/xml-handouts.pdf 090217

[3] http://en.wikipedia.org/wiki/HyperText_Transfer_Protocol 090217

[4] http://en.wikipedia.org/wiki/GUI_widget 090217

[5] http://en.wikipedia.org/wiki/SQL 090205

[6] http://en.wikipedia.org/wiki/Javascript 090205

[7] http://en.wikipedia.org/wiki/Integrated_development_environment 090217

[8] http://en.wikipedia.org/wiki/Subversion_(software) 090205

[9] http://www.danube.com/scrumworks 090217

[10] http://www.danube.com/ 090217

[11] http://www.eclipse.org/equinox/ 090217

[12] http://www.jeffsutherland.org/oopsla/schwapub.pdf 090218

[13] http://en.wikipedia.org/wiki/Scrum_(development)#History 090218

[14] http://jetty.mortbay.com/jetty/ 090222

[15] http://www.eclipse.org/eclipse/ 090223

[16] http://Qooxdoo.org/about 090225

[17] http://www.osgi.org/download/r4v40/r4.core.pdf p. 27 090217

[18] http://www.osgi.org/About/Technology 090210

[19] http://en.wikipedia.org/wiki/Rich_Internet_application 090205

[20] http://en.wikipedia.org/wiki/Java_Database_Connectivity 090210

[21] http://code.google.com/intl/en-EN/webtoolkit/ 090218

[22] http://en.wikipedia.org/wiki/API 090217

[23] http://www.aptana.com/docs/index.php/About_the_Eclipse_Workbench 090217

[24] http://en.wikipedia.org/wiki/Scrum_(development) 090206

[25] http://en.wikipedia.org/wiki/User_story 090206

[26] http://www.eclipse.org/articles/Article-RCP-2/tutorial2.html 090302

[27] http://wiki.eclipse.org/index.php/Rich_Client_Platform 090302

[28] http://en.wikipedia.org/wiki/Graphical_user_interface 090303

[29] Fabian Lange. *Eclipse Rich Ajax Platform: Bringing Rich Client Into the Web*. Apress, 1st edition, 2008

[30] http://en.wikipedia.org/wiki/Drag-and-drop 090303
[31] http://www.macromedia.com/software/flash/about/ 090304
[32] http://www.rap-book.com/ 090304
[33] http://www.mysql.com/ 090309
[34] http://en.wikipedia.org/wiki/Subversion_(software) 090309
[35] http://en.wikipedia.org/wiki/File:Scrum_process.svg 090309
[36] http://en.wikipedia.org/wiki/Single_responsibility_principle 090317
[37] http://en.wikipedia.org/wiki/Singleton_pattern 090317
[38] http://en.wikipedia.org/wiki/Identity_map 090317
[39] http://www.eclipse.org/jdt/overview.php 090322
[40] http://www.eclipse.org/rap/about.php 090322
[41] http://java.sun.com/j2se/1.5.0/docs/api/java/util/ArrayList.html 090323
[42] http://azureus.sourceforge.net/ 090323
[43] http://en.wikipedia.org/wiki/BitTorrent_(protocol) 090323
[44] Henrik Kniberg. *Scrum and XP from the Trenches*. InfoQ, 1st edition, 2007
[45] http://en.wikipedia.org/wiki/Planning_poker 090329
[46] http://en.wikipedia.org/wiki/Many-to-many_(data_model) 090330
[47] http://en.wikipedia.org/wiki/Cross-compile 090406
[48] http://en.wikipedia.org/wiki/JUnit 090415
[49] http://en.wikipedia.org/wiki/Context_menu 090416
[50] http://en.wikipedia.org/wiki/XML 090417
[51] https://www.gpublication.com/eclipse/#requestedContent=contentID://EclipseConferences/EC2009/330 090417
[52] http://code.google.com/intl/sv-SE/webtoolkit/doc/1.6/ReleaseNotes_1_6.html 090422
[53] http://code.google.com/intl/sv-SE/webtoolkit/tutorials/1.6/compile.html 090423
[54] http://www.eclipse.org/mylyn/ 090428
[55] http://www.bugzilla.org/ 090429
[56] http://www.atlassian.com/software/jira/ 090429
[57] http://office.microsoft.com/en-us/communicator/HA102037151033.aspx 090507
[58] http://en.wikipedia.org/wiki/LAN, 090514
[59] http://tomcat.apache.org/, 090605

A Details of the database tables

Products:

	ProductID	Description	Date	Closed
Description:	The unique id of a product	A description of the product	Represents the creation date of the product	Represents if the product is finished
Data type:	Integer	Longtext	Datetime	TinyInt
Allow null:	No	No	No	No

Keys

Candidate	ProductID
Primary	ProductID
Foreign	None

Users:

	UserID	Name	Email	Pass	systemRole
Description:	The unique id the user	The name of the user	The email address to the user	The users' password	ID representing the system role
Data type:	Integer	Text	Text	Text	Integer
Allow null:	No	No	No	No	No

Keys

Candidate	UserID, Email
Primary	UserID
Foreign	RoleID

Roles:

	RoleID	Name
Description:	The unique id of the role	A short name describing the role
Data type:	Integer	Text
Allow null:	No	No

Keys

Candidate	RoleID
Primary	RoleID
Foreign	None

Stories:

	StoryID	Name	Description	Benefit
Description:	The unique id the story	The name of the story	A more in deep text describing the story	The benefit of having this story
Data type:	Integer	Text	Text	Integer
Allow null:	No	No	No	No

	Penalty	ProductID	SprintID
Description:	The penalty of not having this story	ID representing which product a story belongs to.	ID representing which sprint the story belongs to.
Data type:	Integer	Integer	Integer
Allow null:	No	No	Yes

Keys

Candidate	StoryID
Primary	StoryID
Foreign	ProductID

Tasks:

	id	Name	Points	PointsLeft
Description:	The unique id of the task	A short name describing the task	Indicates how many points this task will take to implement	Points left until the task is done
Data type:	Integer	Text	Integer	Integer
Allow null:	No	No	No	Yes

	Status	UserID	dateCompleted	StoryID
Description:	Status of the task: 0 = not checked out, 1 = checked out, 2 = done.	Reference to the user committed to finish the task	The date of the completion of the task.	ID indicating which story a task belongs to.
Data type:	Integer	Integer	DateTime	Integer
Allow null:	No	Yes	Yes	No

Keys

Candidate	TaskID
Primary	TaskID
Foreign	StoryID

Sprint:

	SprintID	StartDate	EndDate	ProductID
Description:	The unique id the sprint	Date indicating when the sprint starts	Date indicating when the sprint ends	ID indicating which product a sprint belongs to
Data type:	Integer	DateTime	DateTime	Integer
Allow null:	No	No	No	No

Keys

Candidate	SprintID
Primary	SprintID
Foreign	ProductID

ProductUserRows:

	ID	ProductID	RoleID	UserID
Description:	Unique identifier	ID which references to a product	Reference to which role the user has in the product	ID which references to a user
Data type:	Integer	Integer	Integer	Integer
Allow null:	No	No	No	No

Keys

Candidate	ID
Primary	ID
Foreign	UserID, ProductID