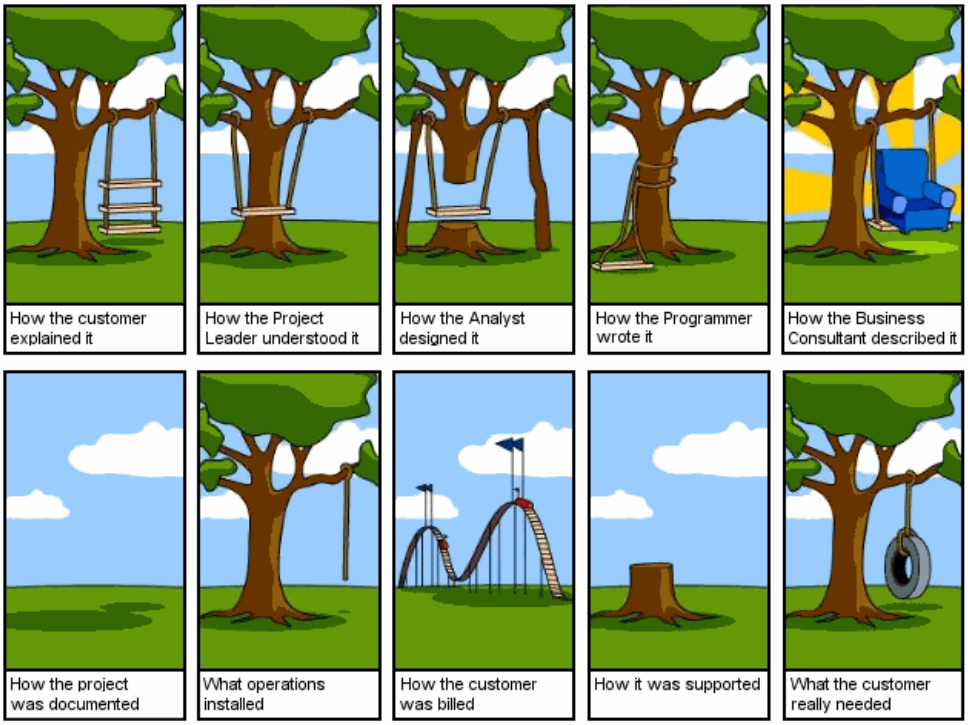
Usable and Adaptable Business Software

Master thesis report



Delft University of Technology
Faculty of EEMCS
Man-Machine Interaction Group

by Leon van der Ree
September 2008

Usable and Adaptable Business Software

submitted in partial fulfilment of the
requirements for the degree of

MASTER OF SCIENCE

in

MEDIA AND KNOWLEDGE ENGINEERING

by

Leon van der Ree
Born in Dordrecht, the Netherlands

Man Machine Interaction Group
Department of Media and Knowledge Engineering
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.mmi.tudelft.nl

T. E. Johnston & Co (Holland) b.v.
Vrijenburglaan 59
Barendrecht, the Netherlands

cover picture from http://www.projectcartoon.com

Usable and Adaptable Business Software

Author: Leon van der Ree
Student id: 1100300
Email: leon@fun4me.demon.nl

Abstract

Background: To be able to adapt to the fast changing markets, companies require flexible systems to support them in their daily business. Instead, most Enterprise Support Software is made for a specific task, not designed to match the needs of a specific company, let alone adapt to changing requirements. To test if this can be changed, the question in this project is: *Is it possible to simplify the development of business software, without being restricted in possibilities, to be able to create custom business software that is usable and adaptable?* In this project a solution will be developed that will be used in a case study in order to answer this question.

Case study: T. E. Johnston & Co (Holland) b.v. is a small company active in the harbour of Rotterdam. They are a cargo superintendent company, supervising the surveys of transshipments of edible oils for their clients. They outsource tasks to external surveyors as well as laboratories, collect their results and combine these in a survey-report, containing quantity and quality checks. To support them in this, they use standard office applications and paper based records to maintain overviews and create reports. This requires them to enter the same data repetitively in several documents. They are therefore looking for an application that can reduce this overhead and prevent them from making errors. Since there are no existing solutions to support them in their activities, they would like to take part in this project and see if a supporting application can be created. Their work process was being investigated and in consultation with them requirements for a support system were set up. Based on this a design was made that, after approval, was implemented.

Methods: Two requirements that companies commonly set are support for their workflow when managing their entities (like products and clients) and keeping an overview of these. Tools and solutions were searched to fulfil these, while allowing for a reduced development time and improved maintainability, so current and future specifications of a company can more easily be incorporated. A technique called 'administration generation' was found, that introduces an abstraction layer to reduce the repetitive tasks seen during development. Unfortunately no implementation of it was found that completely matched all the requirements. Therefore it was decided to use the fully pluggable PHP framework Symfony, in combination with the very flexible JavaScript framework Ext JS. Symfony has an implementation of the 'administration generation'-technique and allows to extend it to make it match the requirements, while Ext JS is integrated to improve the user experience. Both Symfony and Ext JS are known because of their good documentation and active community, which turned out to be very valuable.

Results: At the moment a plugin for Symfony has been released as open-source software and is already used within several companies to support them with their business. It is capable of generating detailed edit- and general overview-screens of a company's data, which for T. E. Johnston & Co (Holland) b.v. means they now only have to enter data once to automatically get overviews and be able to generate reports based on their requirements. The adaptability has already proven itself, when after testing adjusted requirements could be implemented within a day.

Thesis Committee

Chair:	Prof. Dr. Mark Neerincx
Advisor:	Dr. Stijn Oomes
Committee Member:	Dr. Eelco Visser
Committee Member:	Dr. Ir. Willem-Paul Brinkman

Preface

This project has been performed to investigate if custom software for (small) companies can be created affordably, while keeping it flexible. This way it can change according to possible new requirements in the future and really support a company.

To accomplish this a technique called administration generation has been extended to make it possible to generate entire screens for editing representations of their entities or getting overviews, with the least amount of code. An application has been designed for the company T. E. Johnston & Co (Holland) b.v. as a case study to see if it really is possible to generating usable applications with the help of administration generation.

Since this project is fairly large I have decided to make my work open source at an early stage of development. This way I wanted to get support from online communities for tips, for help with implementing improvements and for testing. As a result the plugin is now not only being used for the project at T. E. Johnston & Co (Holland) b.v. to support their activities, but developers from other companies have already been using and improving this work, among which the Citi-group[1] and a new company with similar ideas called Codebase[2].

For this project I made used of open source frameworks and got some valuable feedback from community members, who I hereby want to thank. In particular I want to thank some members from the Symfony community:

- Wolfgang Kubens who initiated the idea, created the early base from which I could kick-start the project and for his work on the ExtJs2Plugin.
- Andre Schuurman (DrCore) who provided the base to retrieve foreign values.
- Jérôme Macias who helped me structure my code and has rewritten a big part of my code in the early stages of development.
- Benjamin Runnels (KRavEN) working at the Citi-group, by far the biggest contributor: constantly using, testing and extending this plugin and comes with great ideas and accompanying projects.
- Lukas Meyer from codeBase GmbH, who also used and tested this plugin, added some improvements especially regarding i18n and now provides resources for documentation.

And from the Ext JS community:

- Jozef Sakalos (Saki) for proving great insight in how to develop nice applications in JavaScript/ExtJS
- Doug Hendricks for his work on the Basex library and Managed-IFrame
- Andrew Mayorov for his work on the TinyMCE integration in Ext JS

Last but not least I should of course mention Jack Slocum together with his team for providing the great Ext JS framework and François Zaninotto, Fabien Potencier and their team for providing the great Symfony framework.

Finally I want to thank Matthijs Blaas, Arjan Zijderveld and Wouter van der Graaf, who were always available to provide tips and feedback. And my advisor Stijn Oomes for his guidance and providing me the opportunity to do this project in the first place.

1 http://www.citi.com
2 http://blog.codebase.ch/?p=31

Table of Contents

1. Introduction

After my literature study about Enterprise Support Systems [Ree, 2007], I found out that many companies try to adapt to or work around the functionality of a software application, instead of the software was being adapted to the company [Wallace & Kremzar, 2001] [Hedman & Borell, 2003] [Davenport, 1998]. Companies working with commercially, off-the-shelf applications (COTS) are hardly ever customising it to their specific needs because of the high costs [Themistocleous, Irani, O'Keefe and Paul, 2001]. Companies for which no COTS-solution exists can probably only use standard office applications like spreadsheet and word-processor software to support them in their activities. COTS-applications also make it more difficult for a company to change its business process because of their inflexible nature [Themistocleous, Irani, O'Keefe and Paul, 2001] [Davenport, 1998]. These facts of software not being able to fully support a company introduces all kind of problems, like overhead because of unnecessarily repetitive tasks and an higher error probability [Bouwman, Hooff, Wijngaert & Dijk, 2005] [Laudon, 2002]. This results in companies being far less efficient than desired and losing a possible competitive advantage [Rouse, 1993].

This research project has as goal to see if it is possible to create customised usable business software, developed to suit the needs of a specific company, by supporting their workflow, while keeping costs relatively low. When companies are being supported by an application, no more workarounds should be required, which allows them to focus on their main activities and improve their efficiency. To make it possible to create customised business software, generic business requirements will be searched and used as a base to find tools and techniques that make it easier to implement solutions to fulfil these. A case study will be performed on a company called T.E. Johnston & Co (Holland) b.v. to test if the identified tools and techniques can really assist in delivering a usable application that will support the company.

1.1 Business Software

Based on my literature survey [Themistocleous, Irani, O'Keefe and Paul, 2001] [Bouwman, Hooff, Wijngaert & Dijk, 2005] [Wallace & Kremzar, 2001] [Johannesson & Perjons, 2001] [Aiken, 2002] [Lee, Siau & Hong, 2003], my experiences at different companies and the requirements of T.E. Johnston & Co (Holland) b.v., it appears that companies primarily desire a possibility to edit a representation of the entities (like clients and products) they work with, with properties they work with. They want to reuse these representations in their reports and in overviews, again adapted to their specific business [Rouse, 1993]. Besides, they would like to have the possibility to exchange information from their system with other companies and some standard applications (for example a bookkeeping application they are already working with, or a spreadsheet application so they can play with their data themselves in any way they like). Because companies continuously evolve, supporting applications should be able to handle these changes. [Davenport, 1998]

A final observation that can be made is that people want to be able to access their data from multiple locations; so not only from their office but also from their home or when they are in a hotel. This however also introduces some extra needs regarding security, since the applications cannot run standalone or in a local network any more and it is not desired to have the company's data out in the open.

To support a company with their activities an application should be able to:

- Manipulate and process the company's data, supporting their workflow [Lee, Siau & Hong, 2003]
- Maintain overviews of this data
  - Lists of multiple items
  - Detailed per item
- Support communication of this data [Markus & Tanis, 2000] [Lee, Siau & Hong, 2003]
  - to different people
  - to different companies [Feldberg, 2007]
  - to other applications
- Adapt to changes within a company [Reijswoud & Heuvel, 1997]

It would be highly desired for application to be:

- Accessible
  - easy to install (for new employees, on new machines or at temporary locations)
  - easy to use
  - accessible from anywhere, allowing to work from
    - home
    - a hotel
    - on the road
- Secure
  - not accessible for others
  - not showing any data for which no credentials are available

1.2 Research Question

Since almost every company has its own data model and workflow, and since companies often have the tendency to change their requirements over time [Keuning, 2001] [Davenport, 1998], it would be nice to see if the implementation of business applications can be done faster and easier to be able to create applications that can really support a company. To make this possible my goal is to reduce the complexity of implementing business applications without restricting them in their possibilities: making it possible to conform to specific requirements of a company and to be able to incorporate possible new requirements in the future more easily. In the end these custom applications should provide functionality that is not available in COTS-applications.

Therefore the main question that I try to answer in this research project is: *Is it possible to simplify the development of business software, without being restricted in possibilities, to be able to create custom business software that is usable and adaptable?*

To answer this question, information is gathered by trying to implement an application that is

- usable: the company should be able to efficiently do with its application what it requires
- adaptable: it should be simple to adapt the application to fulfil changed requirements

with an iterative and user-centered development style, for a company for which currently no support software is available.

It should be tested:

- if the company approves the application. Since they are the one with the knowledge about the working of the company, they can tell if the application fulfils their requirements.
  → this is done with a user test
- if the implemented code really is less complex
  - is it faster to write
  - is it easy to write
  - is it easy to maintain

  → this is done by comparing the written code
  → and looking at the code that needs to be changed to meet changed requirements

1.3 Approach

To make custom and adaptable business software, I would like to simplify its development process. However, before that is possible, some general requirements for these applications should be identified, after which tools and solutions can be found to fulfil these requirements. General requirements for business software have already been described in section 1.1, but to confirm these and to test if the constructed solution will really work out, a case study will be performed in this research project.

A real company will be used, that will be thoroughly analysed in order to derive requirements for it. Tools will be searched, to construct solutions that can fulfil their requirements and tests will be done to verify if the application is really capable of fulfilling the company's requirements.

To be able to deliver a product that really suits a company's needs (supporting their workflow and manage their data) and in order to prevent miscommunication and misunderstandings as illustrated

on the front-page, a user-centered approach will be used. Frequent communication and several prototype iterations should allow the company to explain their wishes and confirm the implementation meets their expectations. This way the user (who has the knowledge about the working of the company) can get a clear idea about the final application and can get actively involved in the development process to make the application really match their requirements. [Verville, 2003] [Soh, Kien & Tay-Yap, 2000]

1.4 Overview of Thesis Report

In this thesis report I will describe the steps I made to find and verify a solution to reduce the complexity of implementing custom business software. Even though the solution was implemented in several iterations this report will describe the process linearly. As a consequence the designs provided in this report will not completely match the final result. Differences between the design and the final result will be highlighted when applicable.

This thesis concerns a case study of a company called T.E. Johnston & Co (Holland) b.v. and this company will be analysed in chapter 2. From this analysis requirements were established, as described in chapter 3, and a solution was being designed that fulfil these, as described in chapter 4.

In chapter 5 the technical design is described, that contains the base for the implementation. Tools and techniques were searched that could improve the development process for this base. The most important tools and techniques that were found to make this possible are the Model-View-Controller pattern (MVC), Object Relational Mapping (ORM) and Administration Generation. These tools and techniques were found combined in several frameworks and this can all be found in chapter 6.

However none of the tools that have been found provided enough functionality to build the entire application with. Extensions were required to improve both the functionality and the interactivity. It did not take long to realise that this would be quite ambitious, therefore it was decided to make the solution open source. With the power and help of an open source community the solution could become much better and be implemented much quicker. In chapter 7 it will be described how making it open source helped and what extensions were combined into a reusable plugin.

With the plugin the implementation of the support application was relatively easy. In chapter 8 one can find the result of the implementation. This result was tested with the user in order to be able to evaluate its functionality. Even though there was a lot of communication with TEJ during the development, it showed that the initial application was not completely matching the workflow within T.E. Johnston, but the improved development tools made it possible to modify the application very easily. Chapter 9 describes all this.

Chapter 10 contains the overall conclusion of this research assignment and ends with some recommendations for future work to make it possible to further improve the development process.

A glossary can be found at the end of this report, in which the abbreviations and terminology get explained.

2. Case study: T. E. Johnston & Co (Holland) b.v.

The first step when developing a company centered application will be to analyse the company the application is intended for. T. E. Johnston & Co (Holland) b.v. (from now on called TEJ) will be a good case for this project: at the moment TEJ does not have a support system, because their activities are not common enough to be implemented in COTS-software. Instead almost all of their activities are being performed with the help of office applications, in which they have to perform many repetitive tasks. Creating a supporting application for them can show some requirements needed to build custom business software and confirm that creating usable software is possible.

2.1 Methods

The analysis of TEJ is done with the use of several techniques.

At the beginning an interview at TEJ was held to discover their problems and wishes. This was followed by an analysis of their documents, like reports and overviews, to create an image of the company. Additional interviews were held to understand the working of the company. The information gathered from these interviews and documents was combined in a wiki[3], that was checked and updated by TEJ.

To understand the workflow at TEJ observations were made at the office, during a normal workday. It showed how tasks are being performed and most notably, how they were often interrupted by phone calls from clients and other related parties to exchange information.

The results of this analysis are described in this chapter.

3 http://fun4me.demon.nl/wiki/afstuderen/index.php/TEJ

2.2 Overview

T. E. Johnston & Co (Holland) b.v. is the Dutch division of T. E. Johnston & Co. The company is located in Barendrecht, where two employees are supervising surveys for their clients. These clients provide an assignment with a request to check the quantity and quality of their edible oils during loading, discharging and transshipments in the harbour of Rotterdam and vicinity (See figure 2.1). This results in an official survey-report, containing a log of the transshipment and the precise information about the quality and quantity of the products, that the clients use for the final payment of their client.

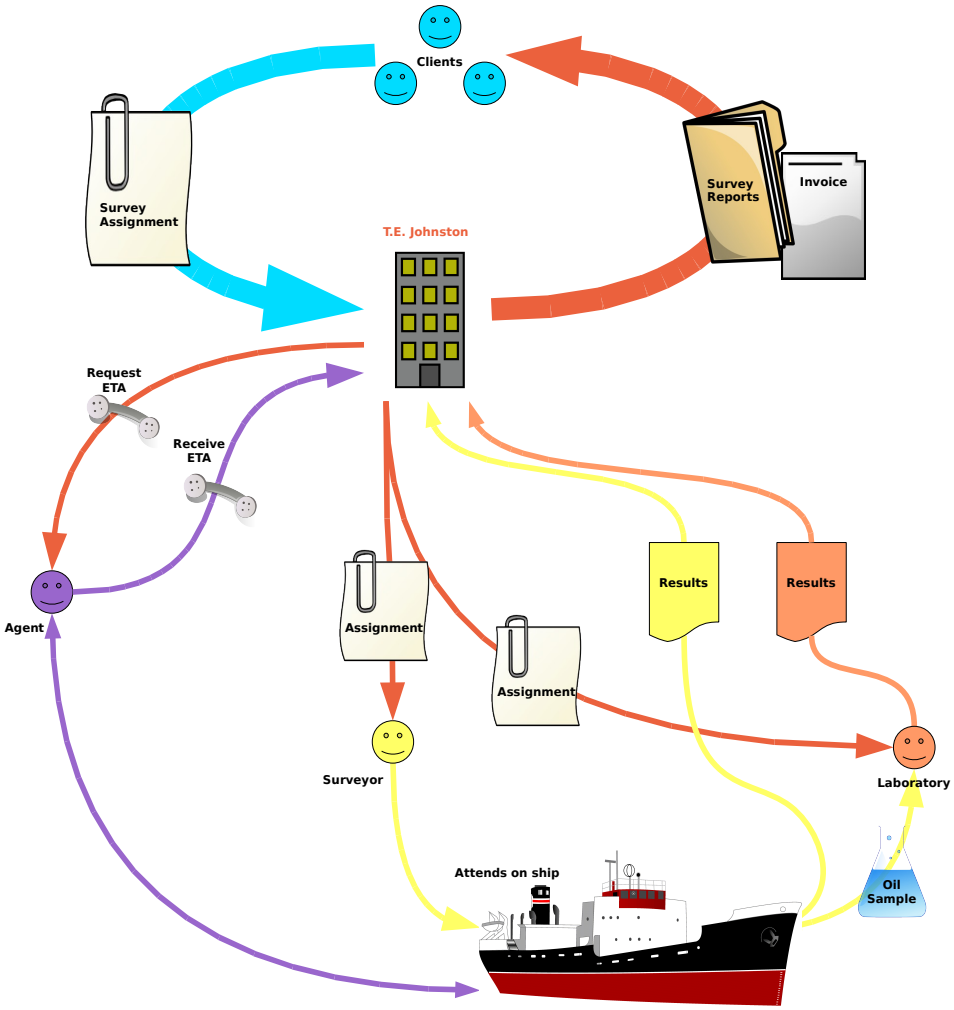


Figure 2.1: Overview of the main business of T. E. Johnston & Co (Holland) b.v. together with the main actors involved

During the surveys TEJ keeps in contact with several different parties to manage the process. The surveys are being performed by externals parties (surveyors) on behalf of TEJ and sometimes additional analyses are being requested that are performed by an external laboratory. To arrange these surveys, the estimated time of arrival of the vessels is being checked, by contacting the agents of the vessels. Assignments for both the surveyor as well as the laboratory are created based on all this information. In the meantime clients are constantly kept up to date by phone and small e-mails.

Computers are used more and more, not only to create the final reports and to maintain an overview of the current inspections being done, but also as a communication medium. The fax for example is rapidly getting replaced by e-mail. A client-server architecture has been implemented, so both documents as well as e-mail are now centrally organised and accessible by all employees. A good internet connection and protocols like IMAP, FTP and SSH make it possible to get access to all this information from other locations as well. (For a precise overview of the inventory, see appendix A.)

Figure 2.2: The office at T.E. Johnston & Co (Holland) b.v.

2.3 Actors

While performing their tasks, TEJ gets into contact with several different parties. In this section a short overview of all these parties, including TEJ themselves, will be provided.

T. E. Johnston & Co (Holland) b.v.

T. E. Johnston & Co (Holland) b.v. is the focus of this project. They want to improve their efficiency by getting an application that can support their activities. At TEJ two persons are responsible for managing the surveys and creating the reports.

T. E. Johnston & Co (Marine) Ltd.

T. E. Johnston & Co (Marine) Ltd. is the head office of T. E. Johnston, situated in Liverpool, England. Most clients pay to Liverpool and Liverpool pays Holland. At Liverpool they might want to have a system like this in the future as well, but at the moment work similarly to the current situation in Holland.

Clients

Clients are one of the most important actors for TEJ. Clients provide assignments with requests to survey their products, that are (usually) shipped to the harbour of Rotterdam.

- Provide assignments
- Receive reports
- Pay invoices
- Can have their own preferences, like requiring extra analysis

Two different kind of clients can be distinguished.

- **Shippers**
  Shippers sell a product, they are especially interested in the quantity.
- **Receivers**
  Receivers buy a product, they are also very interested in the quality of the products.

Surveyors

Surveyors are working in close cooperation with TEJ. During transshipments there are usually two surveyors on a ship: one representing the receiving party (receivers) and one representing the selling party (shippers). Together they measure, take samples and seal the oil sample. The surveyor of the receiver takes care of calculation of the SG (the Specific Gravity of the oil, which is a ratio of the density at a certain temperature and pressure), from which the final quantity will be derived in order to calculate what needs to be paid. The surveyors send samples for TEJ to a laboratory for quality checks requested by the client.

- Get assignment from TEJ
- Supervise the loading/discharging at the ship
- Take samples
  - Store samples
  - Send a part of the sample to the laboratory
- Send results to TEJ

Laboratory

The laboratory analyses samples in order to check the quality of the products.

- Receive assignment from TEJ
- Get samples directly from surveyor
- Send results to TEJ

Agents

Agents are assigned by shippers and provide a way of communication between the shipper and the people at land.

- Agents are not known by the client, but are usually found by own knowledge. Else the client or terminal will be called
- Call to receive the latest Estimated Time of Arrival (ETA) of a ship
- Has to receive the B/L (Bill of Lading, which is a bond representing the value of a product) before the ship has entered the harbour.
  - Sometimes the B/L is send directly from the client to the agent (without TEJ in between)
  - Sometimes when the B/L hasn't arrived yet a copy of the B/L is shown to authenticate, this copy is called an LOI (Letter Of Indemnity)

Terminal

Terminals are the location where the ship's cargo gets transshipped.

- Terminals can be called to find the agent of a ship
- They can call for the B/L-numbers

Insurance

The insurance companies sometimes get called in case of problems. This is arranged in cooperation with the client and the surveyor.

- Insurance is called in case of problems
- They investigates problems
- They keeps TEJ and the client up to date of their findings

Courier

TEJ makes use of couriers to send valuable papers, or papers with high priorities.

- The couriers transports the B/Ls to agents
- They transports reports and invoices to clients

Accountant

One last actor who plays a role in the process at TEJ is the accountant. The accountant assists with bookkeeping. However, since bookkeeping needs to comply with government regulations, it will not be a part of the application, and so does the accountant.

- The Accountant assists with bookkeeping

2.4 Workflow

As can be seen in figure 2.1, clients provide TEJ with assignments to survey their parcels on a vessel. TEJ supervises these assignments by creating assignments for third parties (like surveyors and the laboratory) and at the end delivers a survey report contain information about the quality and quantity of the products.

In some cases it can happen that a client announces he can sell his parcels before the vessel has entered the harbour. In those cases TEJ will only keep track of the vessel without asking about the parcels until just before the vessel will enter the harbour (see the example in the middle of figure 2.3). Than TEJ will contact the client to find out if the parcels have already been sold or if a survey is required after all.

Instead of assignments without any parcels related to them, it can also happen that several clients have different parcels on a same vessel (see the example at the top of figure 2.3). In some exceptional cases it can also happen the two clients request a survey for the same parcel, where one client is the shipper, the other the receiver. This has no consequence for the tasks for TEJ.

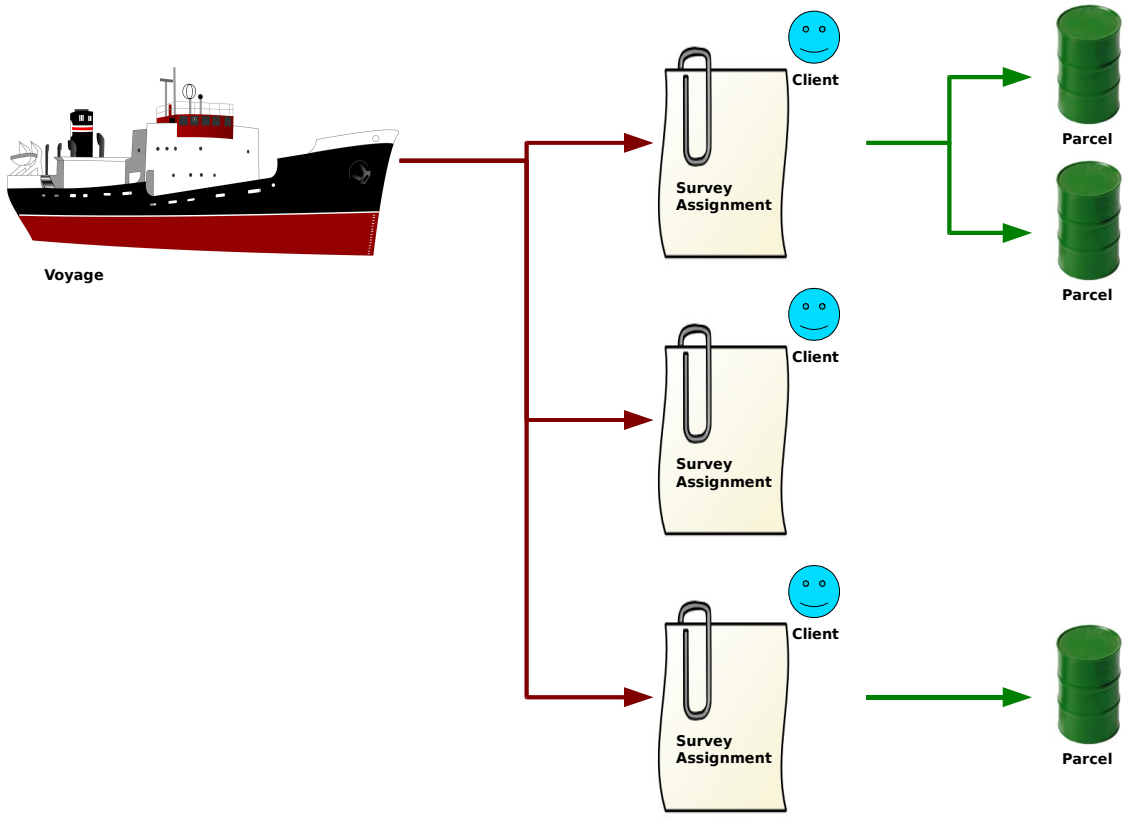


Figure 2.3: An example of how parcels can be related to assignment of different clients, on a vessel

TEJ maintains four different types of overviews of these assignments, used to keep an overview of the tasks needed, to have an archive of all assignments that have been performed and to answer questions about transshipments from different parties. These overviews are:

The Daily Johnston (MS Word)

The Daily Johnston is maintained on the computer in Microsoft Word and contains all assignments that are currently open. That are assignments from vessels that still need to arrive or are currently transshipping. The overview shows properties of the product, client and vessel as can be seen in figure 2.4. Assignments without parcels are also added to the overview, but obviously don't contain information regarding the product. The overview is updated as soon as new information regarding parcels is provided. Agents are called Monday, Wednesday and Friday to update the ETA.

This overview is called the Daily Johnston, since it always contains the latest information, just like a newspaper. It is send daily to the surveyors and Liverpool.

ETA	AGENT		VESSEL	TS	PRODUCT	FILE	BERTH	B/L'S + ANA	CLIENT	STOWAGE
RUNNING	BARWIL	*	ALAM BITARA	10200	CNO	814607 B	KOOLE		Client 1	P1,W6
30/06/07 NOON	RNM	*	STAR BONAIRE	1250	RPS	815607	HAMBURG	BOMINGFLOT SILO	Client 2	
30/05/07 0900	OBC	1*V	NIKOS TOMASOS	1000	RPS	815707	VOPAK	KUA/RTM 28,29 B/L 29 MISSING /VH12	Client 3	P1
30/05/07 1100	RNM	*	STAR ARUBA	500	RPO	815807	IOI		Client 2	
02/06/07	BARWIL		CHAMPION POLAR	500	RPS	813707	KOOLE	SIN-PDG/ROT 05	Client 2	
02/06/07	BARWIL		CHAMPION POLAR	750	PKO	813707	KOOLE	SIN-PDG/ROT 14,15	Client 2	
02/06/07	BARWIL	*	CHAMPION POLAR	10000	RPS	815207 A	KOOLE		Client 1	C1,W3
02/06/07	BARWIL	*	CHAMPION POLAR	1000	RPO	815207 B	KOOLE		Client 1	P 4
03/06/07	OBC		ANAWAN				VOPAK		Client 3	
05/06/07	VOPAK		CAPE BAYRA				IOI/KOOLE		Client 3	

Figure 2.4: Overview of the current Daily Johnston

The third column contains some special symbols:

*	*instructions made*
0-9	*the number of B/Ls*
V	*B/L is send to agent*

Dirty Book (Paper)

TEJ maintains two versions of their Dirty Book. One on paper (see figure 2.5), the other on the computer, both only used within TEJ. The one on paper contains an overview of all the assignments (so both already processed and the ones just opened). The paper version only contains minimal information. Besides, the file number that is assigned to an assignment it contains the vessel-, the client-, the product-name and the quantity.

Figure 2.5: The Paper Dirty Book (since this document contains valuable information, it has been deliberately made unreadable)

This overview has been named Dirty Book at the time the company was still situated in the harbour and people, with dirty hands from the oil, were looking into it, making the paper all dirty. The paper book is still maintained, since it will not crash, is always available and very mobile.

Dirty Book (MS Excel)

The Dirty Book on the computer has been introduced when computers were being used. It is maintained in Microsoft Excel and only contains an overview of assignments that have been processed. Therefore other properties than in the Daily Johnston are relevant, as can be seen in figure 2.6

Ref. #	file close	flokstra	Subject	Date	Place	Tonnage	Product	Client	Into/Ex	Invoice	balance
8000 07	yes		Bunga Melati 4	03-01-2007	Vopak	1.050,000	RPO	Client 2	Star Bonaire	125707	0,000
8001 07	yes		Champion ventura	04-01-2007	Koole	477,000	RPS	Client 2	Star Bonaire	124907	0,000
8002 07	yes		Maria Knutsen	05-01-2007	Vopak	1.194,418	Soya/Corn	Client 4	VH 2	124407	251,342
8002 07	yes		Volharding 2	07-01-2007	Koole	251,342	Soya/Corn	Client 4	Tk 66	124407	251,342
8003 07			CANCELLED								
8004 07	yes		Bow Rio	07-01-2007	Koole	1.199,974	RPO	Client 2	VH 11, 12	125107	1.199,974
8004 07 a1	yes		VH 11, 12	08-01-2007	KWH	1.194,770	RPO	Client 2	Star Aruba	125107	0,000

Figure 2.6: Overview of current Dirty Book

ETA-list (MS Word)

The ETA-list (Estimated Time of Arrival) is the other overview TEJ maintains. It contains an overview of all the vessels that are related to an assignment and that are heading to the harbour of Rotterdam. The overview not only shows the last known ETA, but also the difference with the one before (see figure 2.7). This overview is updated on Monday, Wednesday and Friday and is provided to all clients (and Liverpool) as a service.

VESSEL	ETA	PILOT/ BERTH	PREVIOUS ETA	DIFF.	BERTH	AGENT
BOW SATURN	RUNNING				AVR	ODFJELL
YAMAMAH	RUNNING				KOOLE	HANSEN
BOW PUMA	26-11-06	AM	25-11-06	+1	KOOLE/VOPAK	ODFJELL
DOROUSSA	30-11-06		27-11-06	+3	VOPAK	OBC
JO LIND	30-11-06		29-11-06	+1	VOPAK	OBC
BUNGA MELATI DUA	03-12-06		03-12-06	0		MISC

Figure 2.7: Overview of current ETA-list

As soon as the parcels get transshipped and the survey takes place the Daily Johnston and the ETA-list are being updated. After the survey has been completed and the related information has been received the survey reports for the clients will be created. More precise details about the activities have been described in the next section.

The assignments and reports that are being created by TEJ are not provided in this thesis report.

2.5 Activities

Before the requirements for a supporting application for TEJ can be established, it is task to first analyse the workflow of TEJ in some more detail. In this section all main tasks performed within TEJ are described, to get a more detailed view on their requirements. Diagram are provided to get an idea of the bigger picture.

Activity	Process a new Assignment

Tasks	1. Check if assignment is not already known
	  1. If known
	  1. Check for updated information
	  2. Done
	2. Select a new assignment number
	3. Fill in known information (assignment/parcel)
	4. Done

Figure	2.8

Activity	Process new Information

Tasks	1. Get corresponding assignment
	  1. If no assignment exists regarding this information
	  1. Call client to ask about unknown assignment
	  2. Done
	2. Update information (assignment/parcel)
	3. Done

Figure	2.8 and 2.10

Activity	Answer phone
Tasks	1. Regards an assignment 2. Find assignment 3. Get asked assignment/parcel information (from Daily Johnston) 4. Optionally update assignment/parcel information 5. Done
Figure	2.8 and 2.10

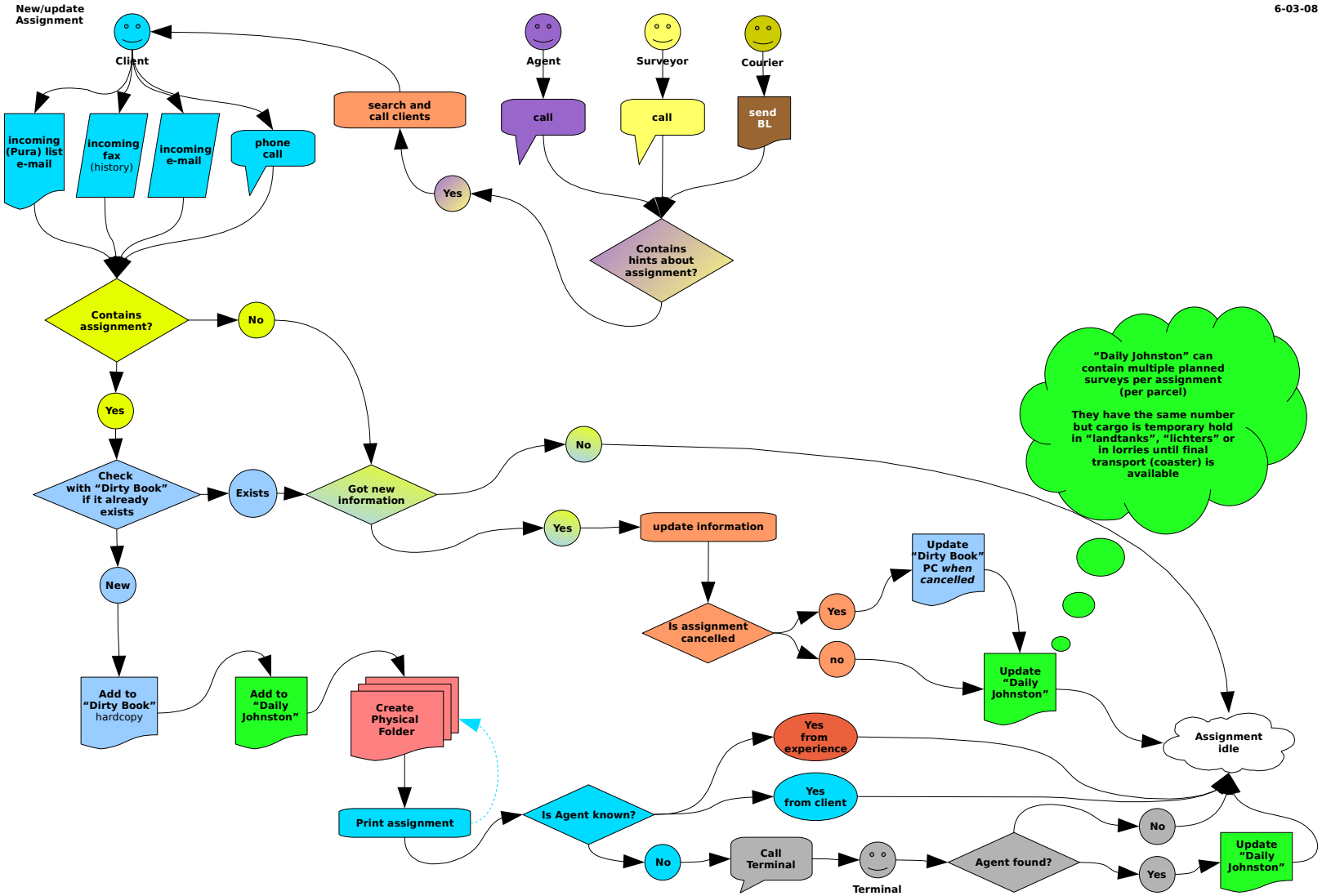


Figure 2.8: Processing information about (new) assignments

Activity	Find Agent when unknown (week before ETA)
Tasks	1. When the vessels estimated ETA is near 2. And Agent is unknown 3. Call terminals to find agent 4. Update Agent (in assignment)
Figure	2.9

Activity	Create Assignment for Surveyor (day before ETA)
Tasks	1. Get assignment of TEJ 2. Transform information to assignment for surveyor 3. Send to surveyor
Figure	2.9

Activity	Create Assignment for Laboratory (day before ETA)
Tasks	1. Get assignment of TEJ 2. Transform information to assignment for laboratory 3. Send to surveyor
Figure	2.9

Activity	Update ETA (mo-we-fr)
Tasks	1. Call Agents 2. Update ETA information (in assignment)
Figure	2.9

Activity	Send ETA (mo-we-fr)
Tasks	1. Get ETA overview of ships to Rotterdam 2. Send overview to predefined group of people
Figure	2.9

Activity	Send Daily Johnston (daily)
Tasks	1. Get Daily Johnston overview of current assignments 2. Send overview to predefined group of people
Figure	2.9

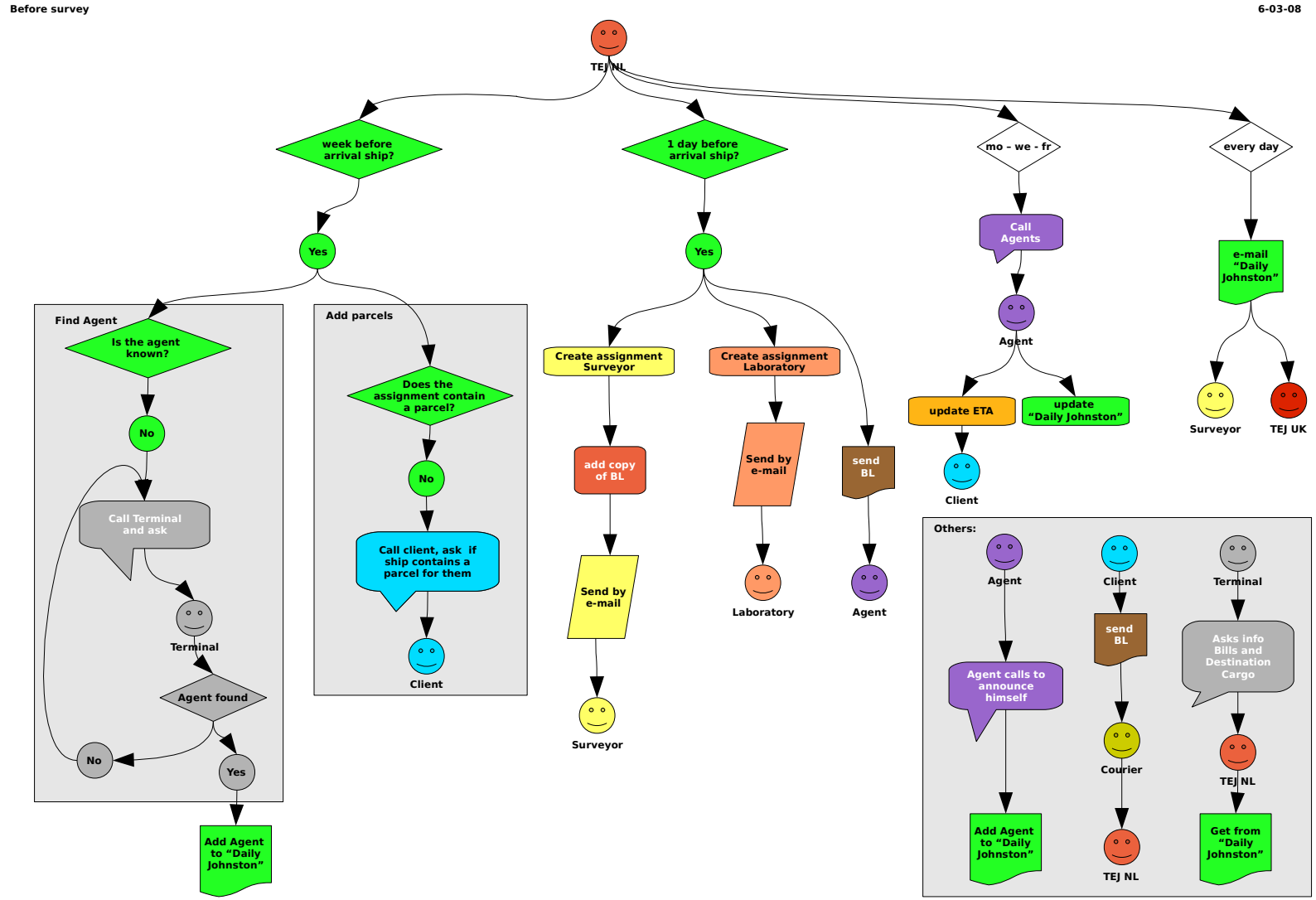


Figure 2.9: Activities before a survey and activities that are regularly scheduled

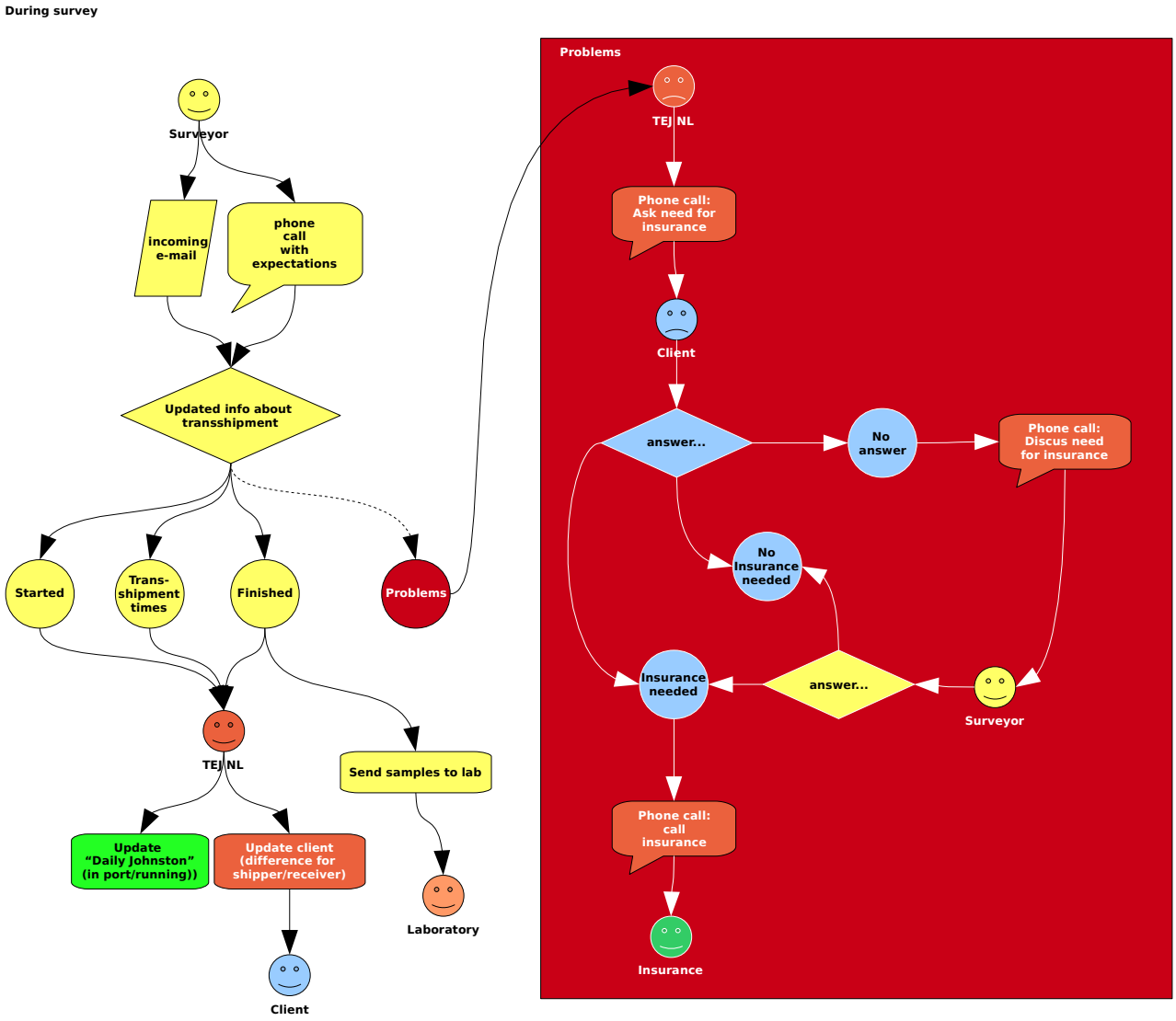


Figure 2.10: Updates during survey

Activity	Create final report
Tasks	1. Get assignment of TEJ 2. Check all information 3. Combine information in report 4. Send to client and TEJ-UK 5. Create invoice 6. Mark assignment as finished, to remove it from the Daily Johnston and make it appear on the Dirty Book
Figure	2.11

Activity	Create Invoice
Tasks	1. Get assignment of TEJ 2. Go to invoice 3. Check calculations 4. Send to client and TEJ-UK
Figure	2.11 and 2.12

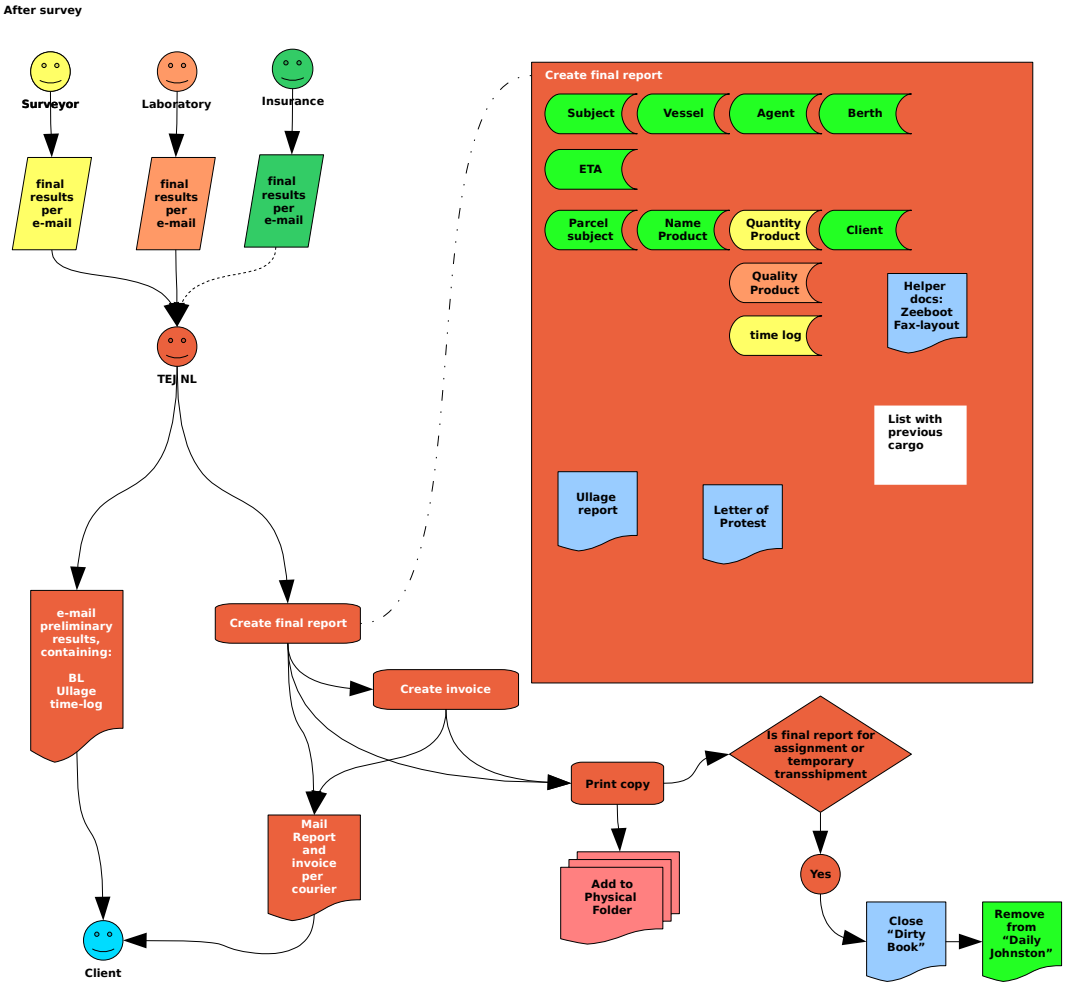


Figure 2.11: Tasks after survey

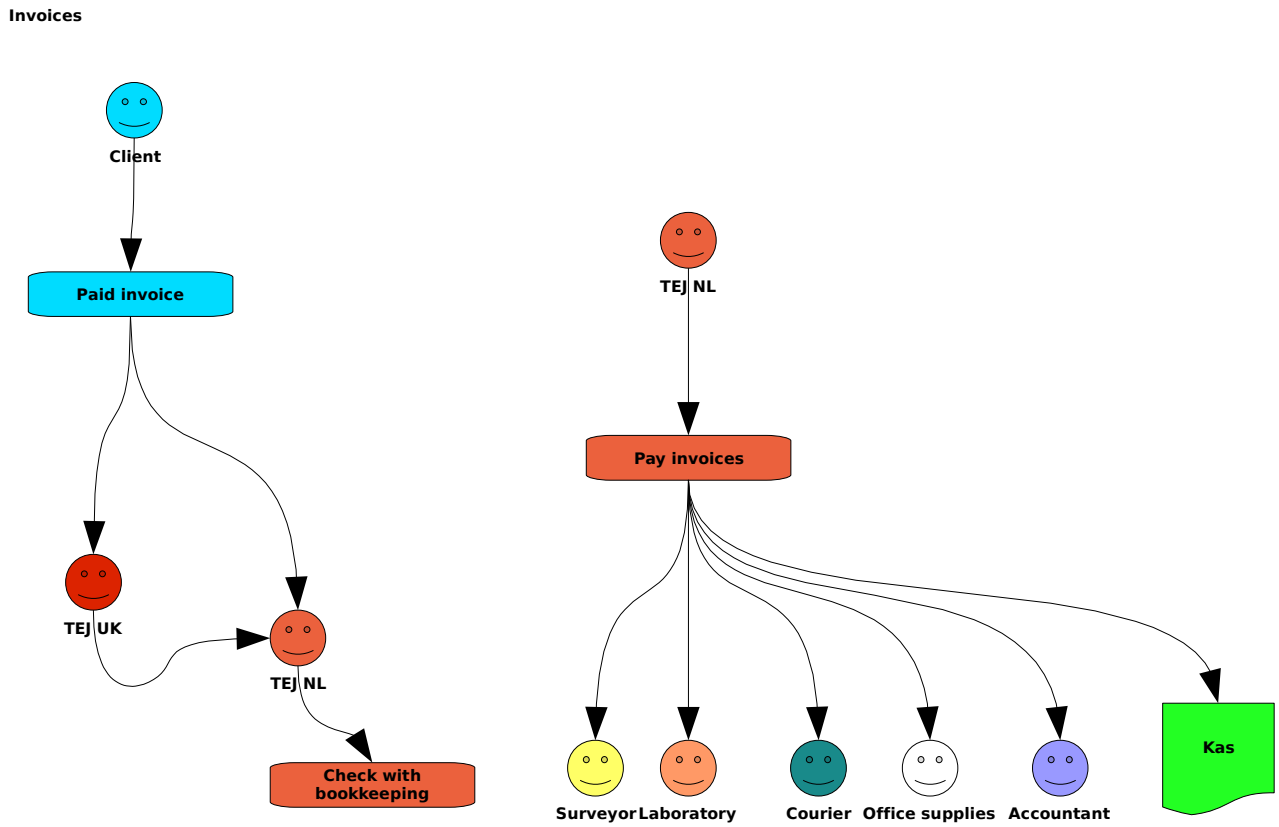


Figure 2.12: Cashflow related tasks

2.6 Problem

Now the business of TEJ is understood and their activities have been thoroughly analysed it is possible to identify and describe their problem.

Even though the activities at TEJ are often very similar, their business is fairly unique. This is the reason why no COTS-software is available to suit their needs and which makes it unattractive for software companies to develop an application only for them.

At the moment TEJ supports their business with the standard office applications Microsoft Word and Excel and the e-mail client Thunderbird. This introduces a lot of overhead, since they perform many repetitive tasks to update all their overviews: The same data is filled in again and again in different documents (from assignments, to reports, to overviews) and at several different places within one document. After all this work has been done, updates are sent to different parties again by manually selecting the documents and sending them to the related companies. It is almost needless to say that besides the overhead this causes, the chance on errors is unnecessarily high.

This however is not the entire story. Computers are depended on power and susceptible to viruses. This makes them not one hundred percent reliable and has introduced a lack of trust in them. Because of this, TEJ also maintains a paper version of the Dirty Book and has a large paper archive containing paper versions of all reports. This obviously further increases their overhead and can only be reduced when their trust gets increased.

3. Requirements

In this chapter the requirements will be described that have been established, based on the information about TEJ, as described in the previous chapter.

3.1 One System

A support application for TEJ should be able to manage their assignments and provide access to their overviews. It should replace the mixture of different office applications, to prevent the repetitive tasks required to update all documents. When all data can be managed with one application, information only needs to be entered once, after which it can be reused at several locations in the assignments and in the different overviews. This will not only prevent making errors, but also allows for spotting and correcting them more easily. [Bouwman, Hooff, Wijngaert & Dijk, 2005]

The support application for TEJ should
- offer access to all data
- require entering data only once

The system should improve the efficiency of the tasks at the company, while reducing the probability of making errors.

3.2 Usable

In order to get the application accepted, it should be usable. To accomplish this, it should support the workflow at TEJ. A clear navigation should allow for easy access to all overviews on the assignments and the information within the assignments should be structured to match the work process. The employees of TEJ should feel comfortable within the application so they trust it. Reliability, security and consistency should aid to this. [Schneiderman & Plaisant, 2005]

The application should support communication with other parties related to a survey. Most of the information that is being communicated is already defined within the system and this can be easily transformed in a message to inform related parties. The quality of the information that is being communicated should be of the same level as in current communication.

When the work process gets interrupted, for example when a client calls to request some information that needs to be looked up, it should be easy to continue with the previous activity.

To become usable, the application should
- provide easy access to all overviews
- match current workflow
- be trustworthy, this can be achieved by being reliable, secure and consistent
- ensure quality
- support communication
- be capable of handling interruptions, allow resuming previous activities

Another functionality that would improve the usability will be to allow easy access to the system from different locations. This makes it possible to work from home or hotels and allows for quick check ups from the road. Besides, communication with other applications like for example bookkeeping applications or generic spreadsheet applications is desired.

3.3 Adaptable

In order to make the development of a usable application like this possible, development tools should provide an adaptable environment [Lee, Siau & Hong, 2003] [Markus & Tanis, 2000]. This environment should support fast and easy maintenance of the application in order to make the development in iterations possible. This makes it possible to change the design during development, to fix problems and incorporate new ideas that have been found during implementation.

It also allows for companies to change the application in the future, to make it match possible new requirements more easily.

4. Design

Now the requirements for TEJ have been established, a design for their support application can be made. The goal of this application should be to reduce the overhead in their current business process, thereby reducing the probability of making errors. It also provides an opportunity to improve the relationship with their clients, by providing them with more detailed information and personalised information.

4.1 Manage Assignment Workflow

To make the application able to efficiently support the workflow at TEJ, the information regarding the assignments of their clients had to be subdivided. This was initially done by separating the information about the vessel and client from the information about a parcel. This way TEJ only had to enter generic data regarding their assignments once, after which they could add the information about one or more parcels to it. During the implementation however it became clear an additional division was necessary to further improve the efficiency of data entry. The information regarding assignments needed to split in information regarding the voyage of a vessel and information regarding the client to make it possible to maintain the ETA-list efficiently. This created a hierarchy matching the overview provided in figure 2.3, where voyages contain one or more assignments, which in their turn contain information regarding the parcels. This division introduced a new concept (of voyages) in the company, but immediately got approved.

Voyages

As said, during the implementation of the application a new division of voyages was recognised and got introduced. Since one vessel can contain assignments for multiple clients it became clear this extra division could reduce the overhead on the input of data support the overview in the ETA-list. At TEJ the properties of a voyage has always been combined in their assignments, but they immediately agreed this extra division was an improvement. Now the ETA of a vessel needed to be updated only once, after which all the underlying assignments of different clients would automatically get updated with this information.

The most important information that is contained by a voyage is:

- The vessel
- The agent of the vessel
- The berth the vessel will arrive at
- The latest ETA of the vessel at the berth
- The assignments

Assignments

The assignments at TEJ are added to a voyage. Assignments contain information about the client and the role of the client in this assignment (shipper or receiver). At the end of the survey the assignment will also hold all invoices, including those of the survey of every parcel. In the case clients provide assignments without any parcels to survey, TEJ will only update the ETA for the related vessel. Once the ETA is near, they will check at the client about the parcels and can add them just before the vessel will arrive at the harbour.

The most important information that is contained by an assignment is:

- The client
- The client-type (shipper or receiver)
- The invoices
- The parcels

Parcels

Parcels will contain information about the product that will be surveyed and provide the main business for TEJ. Every parcel requires surveys during the transshipments and some of the clients require laboratory testing on the product. This all needs to be managed by TEJ. During the transshipments clients will get detailed information about the activities, while also the agents need to be kept informed. At the end of a transshipment the client will receive a detailed report about the activities and the results of the survey as well as the laboratory.

The most important information that is contained by a parcel is:

- The product (with several details like its origin and the load-port)
- The quantity (both estimated, as well as from the "bills of lading", as well as the final measured weight)
- Bills of lading
- Analysis requirements for the laboratory
- Assignment (for the surveyor and the laboratory)
- Results (from the surveyor and the laboratory)
- Reports (for the agent and the client)

Assignments and Reports

During the survey several reports and assignments are being created to support communication, based on the gathered information. Most importantly, the client will get an official report concerning all details of the transshipment, that contains detailed information about the quality and quantity of the product. Besides, the client will be kept up to date with preliminary results as soon as they are available and of course an invoice at the end of the survey. Other documents are created to provide surveyors and the laboratory with their assignment and to inform the agents about the state of the transshipment.

The system can assist in creating these documents, since it already contains all the information that has to be in them. Templates will be used to accomplish this, but the employees of TEJ should remain in full control of the content, to be able to handle exceptional situations.

4.2 Manage Assignment Overviews

Based on the information held by the system about the voyages, assignments and parcels, overviews should be created as currently used at TEJ (see section 2.4).

The four overviews that will be made available are:

- A generic overview containing all assignments, both open and closed
- The Daily Johnston, containing all assignments that are open
- The Dirty Book, containing all assignments that already have an invoice
- The ETA-list, containing an overview of all tankvessels that will enter the harbour of Rotterdam

To allow quick updating of multiple assignments, for example to update the ETAs after calling the agents, it would be nice if some of the columns that contain data which changes often can be changed directly, just like it can be done in a spreadsheet application. However to avoid making changes accidentally it is not desired to have all the columns editable.

4.3 Transition diagram

Based on this information a transition diagram is designed, as can be found in figure 4.1. It shows the entry points the user has for looking at and working with his information. The overviews should be set up to match the current work process at TEJ. So the Daily Johnston will only show assignments and parcels without invoices, where the assignments and underlying parcels in the Dirty Book have at least one invoice linked to them. The ETA-list will only contain information about voyages that still need to arrive. In order to find assignments and parcels for which it is unknown if they contain an invoice, an overview has been added that contains all of them, but with a limited set of information about each item. This overview can be filtered to make it easy to find the intended assignment. All these overviews link to the edit-views to manipulate the representation of the related entity (which can be a voyage, assignment or parcel).

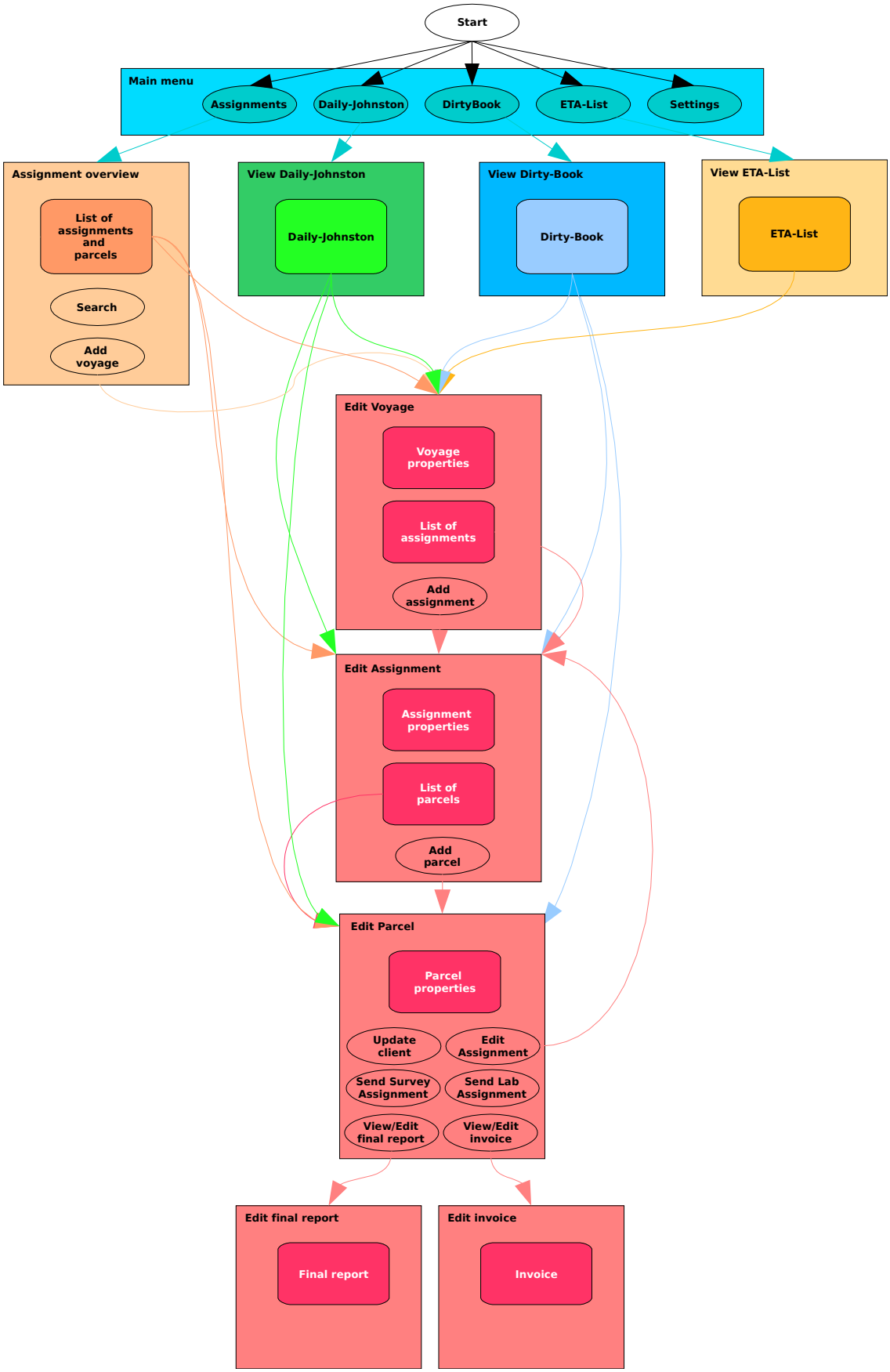


Figure 4.1: Overview of views that should be supported in the application

4.4 Data Model

A lot of information at TEJ is reused within the different tasks and assignments. To make TEJ more efficient data should be entered only once, so it can be reused in all its corresponding views and assignments. To make this possible, a data model is designed that matches to the entities TEJ is familiar with (see section 4.1), and that describes the relations between these entities.

A simplified overview of these representations is described below, for a more extensive view on the data model see Appendix B.

- Voyages
  - ETAs
  - Vessel
  - Berth
  - Agent
  - Assignments
    - Client
    - Invoices
    - Parcels
      - Product
      - Surveyor
      - Laboratory
      - Analysis
      - Bills of Lading

This overview matches with the description as provided in section 4.1. The relations between the entities (that can better visible in Appendix B and will be explained in section 5.2) are not only being used to ensure data integrity (as described in section 6.1), but can also be used to assist the user with his tasks, as will be described in the functional design (section 4.5) under “manipulating data”.

4.5 Functional Design

The new support application should support the workflow at TEJ (as described in sections 2.4 and 2.5) and incorporate all the requirements (as described in section 3). Based on the described analysis the functional design will describe how the workflow can be supported, while fulfilling the requirements.

Support the workflow

To support the workflow at TEJ as good as possible, the fields in the edit-views will be arranged in groups and under different tab-pages to match real life: both in properties of an item as well as in the order in which the information is known in time and added to the system (according to the states mentioned above). Only the most important and distinguishable properties of an entity will be required, other properties are optional and can be filled in at any time, to be as flexible as possible. Also every property can be changed at any time, to correct errors and to give users absolute control to improve its trust in the system.

Take as an example an assignment in which the client only requires TEJ to track a vessel, since he does not yet know if he will sell the products before reaching the harbour. Than TEJ will only add the vessel and client into the system, find out about the agent and after that only keep track of the ETA of the vessel. No information regarding the parcels is required to be added to the system. Once the client informs TEJ about the final information of their parcels, the information can be added per product to the system, or the entire assignment can be cancelled, depending if the products were sold.

State aware edit-views

Since the edit-views will be divided into several tab-pages that match the stages during a survey, states can be maintained to improve the usability of the application. This way the user will automatically see the view, with all the the appropriate fields, that matches the current state of the assignment.

Assignment and parcels can be divided in several states, that are subdivided by several corresponding actions:

States in assignment

1. New assignment
    1. Add general data about the assignment (like the client, the vessel, the berth, etc.)
    2. Add parcels (if known, see below)
    3. Edit ETA of vessel (if known)
2. Update general assignment information
    1. Update assignment with new information
3. Update ETA of vessel
    1. Edit ETA of vessel
4. Vessel arrived
    1. Update ETA timelog data with current action (E.G. running)
    2. Send update to the client
5. When completed
    1. Add invoice
    2. Move assignment and parcels from the Daily Johnston to the Dirty Book
6. Assignment is paid
    1. Close assignment

States in parcels

1. New Parcel
    1. Add data of parcel (like product information and estimated quantity)
2. Update Parcel
    1. Update information of a parcel with the latest and most complete information
3. Before arrival of vessel
    1. Send assignment to surveyor
    2. Send assignment to laboratory (if required)
    3. Send agent a receipt
4. Surveyor results arrived
    1. Add/Update the results of the surveyor (support/check calculations)
    2. Send overview to Client (option to send to others)
5. Laboratory results arrived (if any)
    1. Add/Update results laboratory
    2. Send overview to Client (option to send to others)
6. Create the final report
    1. Write the final report
    2. Send report and invoice (of assignment) to client (option to send to others)

Note: Since the split of assignments in voyage- and client-information was introduced during implementation. Therefore the arrangement of tab-pages, and so stages got arranged differently during the implementation. After some more iterations of development and testing it turned out the client did not had the desire to track the states of an assignment at all. Because of these reasons only the arrangement in different tab-pages has remained.

Manipulating data

Another example of how the workflow is supported, is by the way how new items for their entities (like countries, cities, products, etc.) can be added to the system. Instead of requiring these items to be added to the system in advance, they can be added at the time they are being used for the first time. For example when adding a new assignment, it is not desired to first having to check if the product the assignment is concerned about is already in the system. (And if not, first being required to add it, after which the assignment itself can finally be added for the specific product.) You should be able to define the product during the creation of the new assignment itself. If the product is already in the system auto-completion can be used to give suggestions about the product name, otherwise you can fill in the name of the product and when the new assignment is added to the system, a dialogue should appear showing you the product is not yet known in the system and allowing you to directly add it to the system.

To reduce the cognitive load for the user, data that can be found in the settings overviews (like clients and cities), can also be added and updated directly from the fields where you use them in the assignments and parcels. This way you do not have to add them in advance. For example you do not first have to add the country “the Netherlands” before you add a city like “Rotterdam” to the system, you can simply provide the name for this country while adding this city, the system will ask to add the country to the system automatically to ensure the integrity if it is not known. Because of this the necessity to have overviews on all entities was not very big and therefore these overviews are not available yet

To simplify the manipulation of properties that are changed often or that are updated for many items at once it would be nice to have the opportunity to change these directly from the overviews. For example ETAs are usually updated for multiple vessels after calling agents. To make the updating of ETAs easier it would be nice if you can add their latest ETA to the system directly from the ETA-list, so you will not be required to open every assignment for which a new ETA has been provided.

Overviews and reporting capabilities

Overviews and reports can be generated based on the data that is stored in the system. The Daily Johnston, the Dirty Book and the ETA-list are examples of overviews that are frequently used at TEJ. In the overviews constraints are set to filter information based on the requirements of TEJ.

Reports used at TEJ are for example the surveyor and laboratory assignments and the final reports for the clients. These are clearly also based on the data that is acquired during the assignment and which is already in the system. Templates can be used to define the layout of the reports. Since there can be a lot of exceptional situations at TEJ, the report's content should be fully adjustable in a text-editor after generation.

Navigation

To navigate through all views (as described in section 4.3) a main menu should be visible at all times, providing links to the most important tasks that can be performed within the system. These tasks are:

- View all Assignments (as currently available from the paper version of the Dirty Book)
  - Add new assignments
  - Find assignments already in the system
- View the Daily Johnston (as currently available in Word)
  - Showing all open assignments and parcels
- View the Dirty Book (as currently available in Excel)
  - Showing all closed assignments and parcels
- View the ETA-list (as currently available in Word)
  - Showing all the vessels that will arrive in the harbour of Rotterdam.
- View all entities (as described in the second paragraph of "manipulating data" under section 4.5, this has not been implemented yet)
  - Containing views for all basic data entities (like client-overviews)

To keep the main menu clear, adding and searching of assignments can be done from the view-assignments overview. However links to open assignments and parcels will also be provided from the other overviews, so you can easily see more details for one assignment or parcel from anywhere you encounter them.

Since the employees at TEJ are often called to look things up, an extra list will be available from the main menu that will contain an overview of all assignments and parcels that have already been opened. This way a user does not have to save his work and can easily return to his original work after looking something up.

Access control

The initial system for TEJ should provide access to the employees of TEJ only. All employees will have the same rights within the system, but will be required to login to enter the system. After they have logged in, the system can track which user is working on which assignment and sign reports with their signature. Because of the required login, the system can also be made accessible from remote locations, for example to support working from home or from an hotel.

4.6 Sketches

Based on all information gathered, a prototype has been designed to illustrate how all tasks at TEJ can be supported. Since I did not had all required development tools available at that time, this initial prototype was made with Impress (similar to Microsoft PowerPoint). It allowed me to design an interactive interface relatively quick, to give an impression of the potential functionality. The interactivity was however severely limited, which is the cause of some of the differences with the implementation. The looks of the interface in these sketches are also drastically different from implementation.

The top menu

As described in section 4.5 under navigation, There should be a way to get access to all overviews. The top menu provides an easy to use overview to all the overviews of TEJ.

The three main views on the assignments (one containing all assignments (both open and closed, the formal Dirty Book on paper), one containing only the open assignments (the formal Daily Johnston in Word) and one containing only the closed assignments (the formal Dirty Book in Excel) are accessible from the first three main-entries in the menu. A reference to the ETA-list will be available next to these, which contains an overview on all voyages.

The most right link in the top-menu will provide a link to the overview of all data entities known in the system. And completely left in the top menu you can also see an icon with a number in it, this is the quick-menu, providing an overview on the assignments currently opened for editing.

Support communication

To support communication an overview of contact persons is displayed at the bottom of every screen. The top of this overview shows a list of groups to filter the contact persons on, and at the right are of the overview shows the selected addressees. The selection is based on the default addressees TEJ wants to mail for the activated view, but the user is free to change this selection.

Due to time constraints this feature has not been implemented yet.

Overview of assignments

The initial design for the content of the assignment overview-views was to show the grouped hierarchy of assignment containing parcels, as can be seen in the simplified overview of figure 4.2. However during implementation it turned out this was very impractical. First of all since the information about the assignment was combined as one line of text in the group-header, which made sorting on any of these fields impossible (see section 8.2 for more details about this). Another problem that showed up was the required splitting of assignments on client information and voyage information (as already described in section 4.1). This would require an extra grouping, making the implementation even more difficult.

Because of the problems recognised during implementation, the final implementation of the overview-views (as will be described in chapter 8) will contain overviews almost exactly matching the current overviews used at TEJ as shown in section 2.4, figure 2.4, 2.6. The overview showing all assignments (both opened as well as closed), which was not used at TEJ before, has been implemented differently as designed as well because it was found to be confusing at first. However because the hierarchical overview was not visible any more in the overviews either, a similar but simplified overview of all voyages, assignments and parcels was being implemented. The result will be shown in chapter 8.

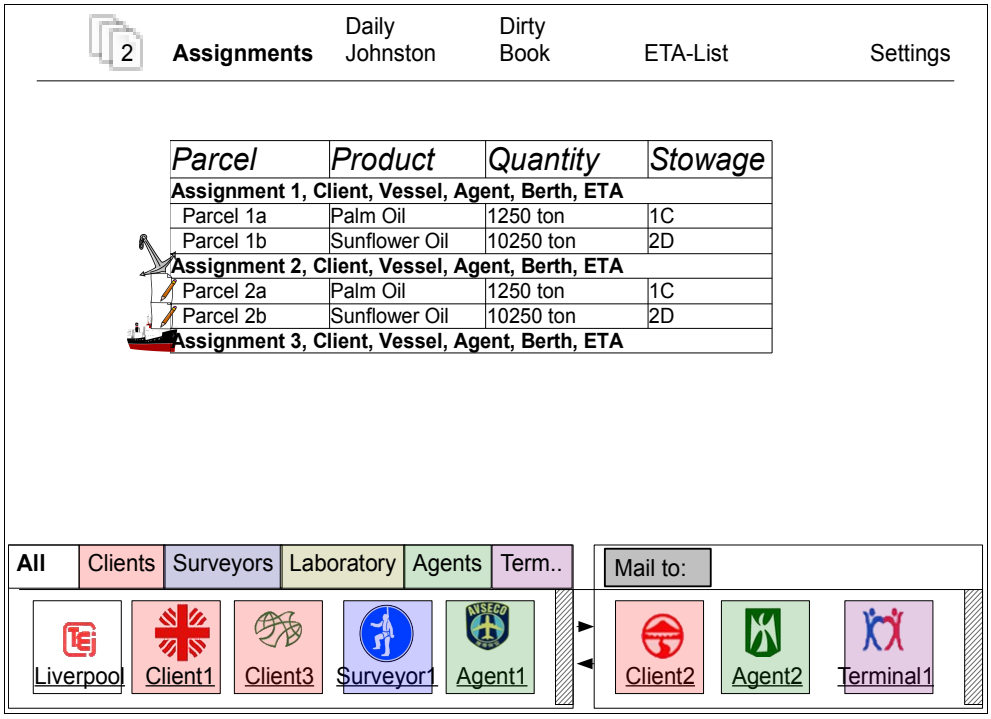


Figure 4.2: Assignment overview, grouping of parcels per assignment

Overview of ETAs

The design for the overview of the ETAs (see figure 4.3 below) exactly matches the currently used ETA-list from TEJ as found in figure 2.7.

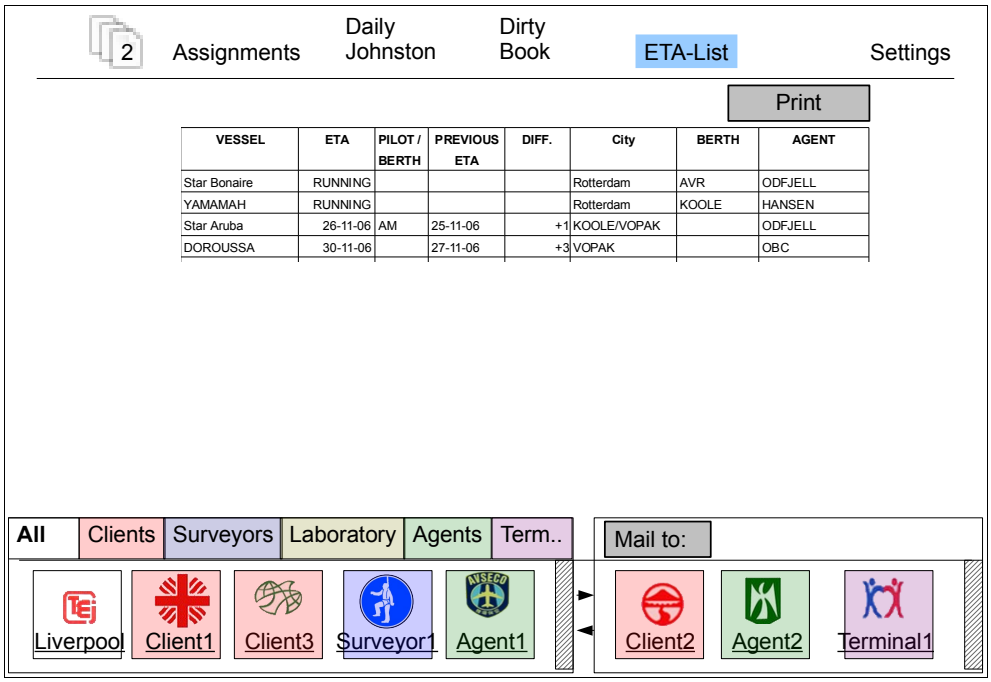


Figure 4.3: Overview of ETAs

Detailed assignment view

As described above, the detailed views are designed to match the workflow at TEJ.

The sketch of the assignment view below still contains a combination of both voyage properties as well as client properties during the designing. At the left side the hierarchy of the assignment with its parcels is being shown. The fields in the content are being divided into several stages as described in section 4.5 under "State aware edit-views". The icons that represent the states, at the top in the content, are filled when a state is fulfilled.

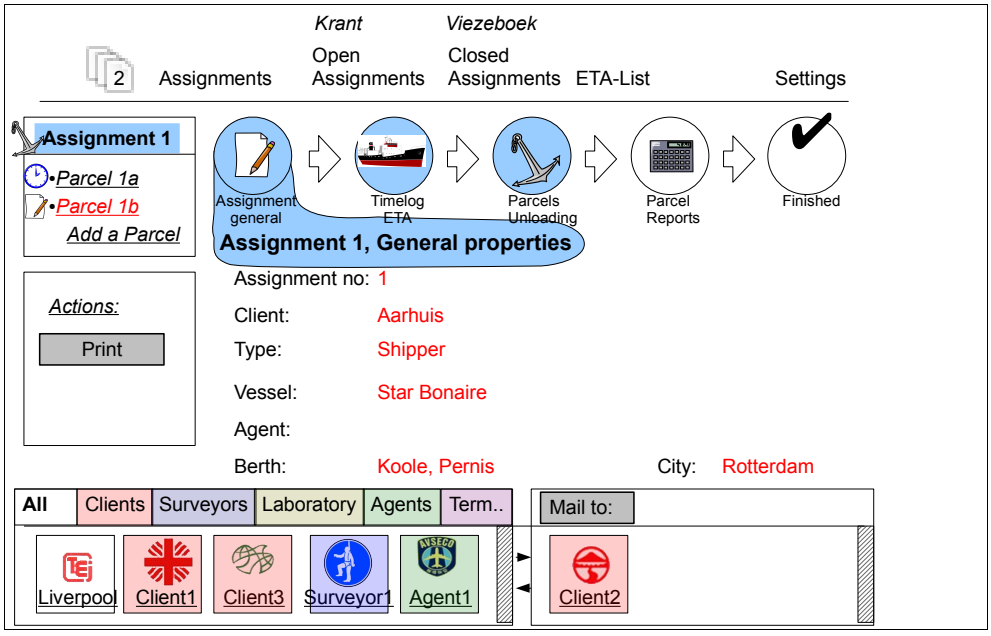


Figure 4.4: Detailed assignment view

This assignment overview is showing the details as a read-only view. There was thought about providing different views for viewing and editing assignments (and parcels), but this introduces some serious issues. The easiest to make different views possible would be to have two entirely different views: one for viewing and one for editing. However the switching between the two would require additional action for the user. Another solution would be to have an intermediate solution, where the fields for viewing could transform to edit fields when the user selected one. This would however introduce some issues for navigation when one wants to edit multiple fields and besides, it would be much more difficult to implement. Therefore all detailed views for assignments and parcels are being implemented as edit-views, as demonstrated in the sketch below.

Detailed parcel edit-view

This sketch shows the states and fields for editing a parcel. In the menu at the left the current parcel is being highlighted. During this design iterations, the states still continuously changed. That is the reason for the "Bill of Lading" state to be temporarily added below the other states.

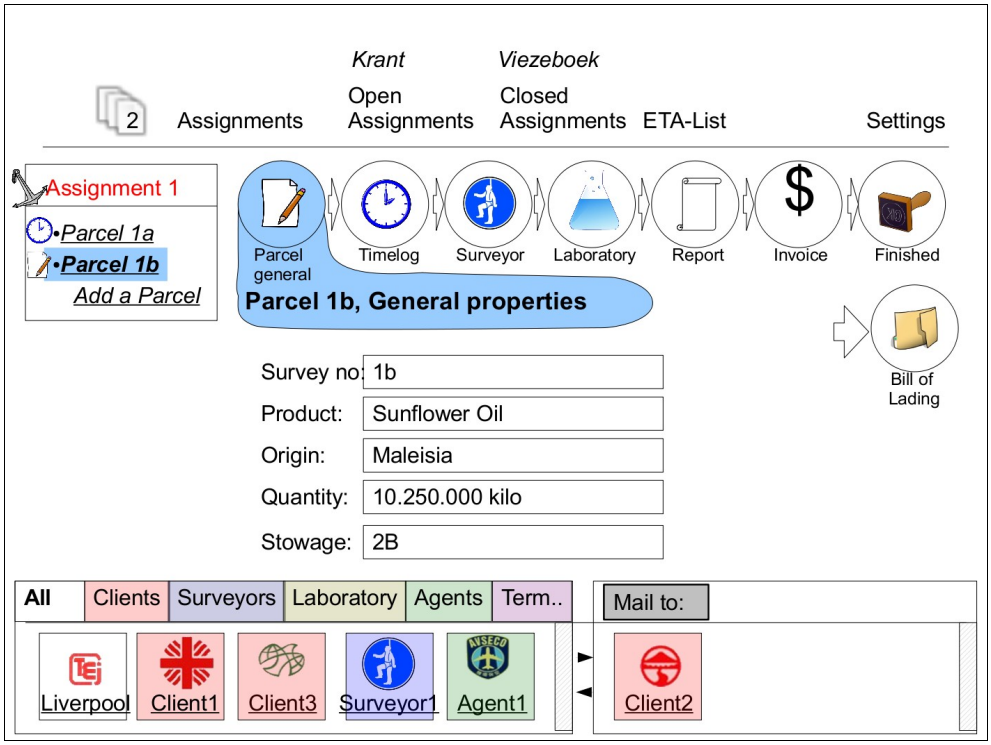


Figure 4.5: Detailed parcel view

Even though there were several iterations during design with approval from TEJ, the invoice state that is shown here under parcels was found to be misplaced during implementation (see section 4.1). This endorses the need for an adaptable development environment.

5. Technical Design

In order to find a way to create custom business software, the design of TEJ will be used to test if the found solution can fulfil their requirements. But the solution does not only need to fulfil their current requirements, it should also be able to fulfil their (unknown) future requirements, as described in my literature survey [Ree, 2007]. This chapter describes the requirements to which such a business application must apply to, what tools already exist to make it possible to fulfil these requirements and how these foreseen requirements are matched.

5.1 Views

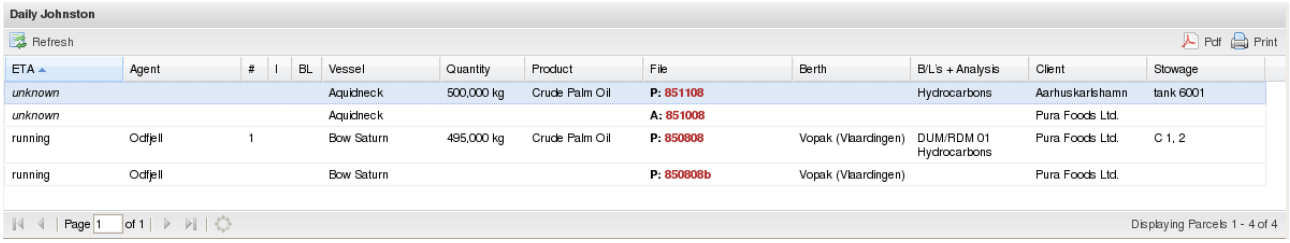
The technical design for the implementation depends heavily on the functional design: make it possible to fulfil the requirements of the user. Therefore the functional requirements for a support application will be inspected and translated into more specific technical requirements.

Lists and detailed overviews of items

When looking at the functional design of the application for TEJ, it becomes clear that there are two main views that return all the time: the overview in lists and the detailed view of items. With these two views almost every situation within TEJ can be represented. It would therefore be a good idea to try to exploit this pattern and reduce the development time.

Lists

Lists contain overviews of entities or sometimes even a combination of several related entities, where every row contains a different item and every column a property of this item (see figure 5.1). For example the ETA-list is a list of voyages and related vessel names, with on every row another voyage and in every column another property of a voyage (like the related vessel-name, but also the latest known ETA, etc.). To make interaction with the list possible, for example to reload its content or print this, actions can be associated with lists.


Daily Johnston

Refresh Pdf Print

ETA	Agent	#	I	BL	Vessel	Quantity	Product	File	Berth	B/L's + Analysis	Client	Stowage
unknown					Aquidneck	500,000 kg	Crude Palm Oil	P: 851108		Hydrocarbons	Aarhuskarlshamn	tank 6001
unknown					Aquidneck			A: 851008			Pura Foods Ltd.	
running	Odfjell	1			Bow Saturn	495,000 kg	Crude Palm Oil	P: 850808	Vopak (Vlaardingen)	DUM/RDM 01 Hydrocarbons	Pura Foods Ltd.	C 1, 2
running	Odfjell				Bow Saturn			P: 850808b	Vopak (Vlaardingen)		Pura Foods Ltd.	

Page 1 of 1 Displaying Parcels 1 - 4 of 4

Figure 5.1: Example of a list

Properties of a list:

- shows an overview of items, divided over rows
- properties of an item (and related items) are divided over columns
- can contain actions

Detailed views

Detailed views show the properties of only one item at a time. This view can be split in two: one where you can only watch the details of an item and one where you can edit the properties of an item. Instead of simply listing all properties below each other, these properties can also be arranged over several tab-pages, grouped in so called fieldsets, split in separate columns or a combination of this all (see figure 5.2). Just like lists, detailed views can also have actions associated with them, for example also to reload their content or to save changes that have been made.

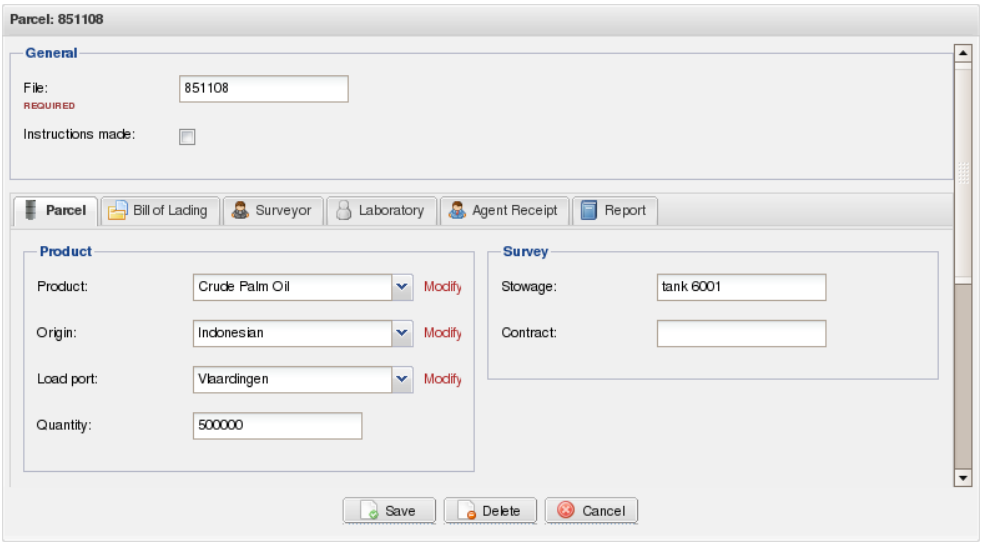


Figure 5.2: Example of a detailed view

Properties of a detailed view:

- show the properties of one item
- the properties of an item can be arranged over tab-pages, fieldsets and columns in their view
- the view can be split in a read-only view and an editable-view
- can contain actions

5.2 Relations

Between the relations among entities, another exploitable pattern can be found. The relations that are seen most often are one-to-many and many-to-one relationships. These relationships are technically exactly the same, the only thing that differs is the viewpoint on the parent object. Since these relations are seen very often and are easy to describe, it makes them a perfect pattern to exploit. However extensive exploits for this pattern at application level have not been found yet. (More about these exploits and their shortcomings will be described in section 6.2.2 and 6.3.2 respectively.)

One-to-many

In one-to-many relationships the parent item refers to many children of (usually) another entity. An example of this for the case of TEJ is for instance a voyage that can contain multiple assignments, or an assignment that in its turn can contain multiple parcels.

One-to-many relationships can be divided in two categories. One as described above where multiple parcels are related to one assignment. In this case every parcel is specifically created (added) for this assignment. This relationship is called master-detail. In the other category a user can select multiple items from a predefined list. For example a product can require a number of standard laboratory checks, these checks can be selected per product from a list of available laboratory checks.

One-to-many relationships have the following property:

- One parent item can get refer to 0, 1 or many children (technically these children will refer to the parent)

One-to-many relationships can be divided into two categories:

1. The related items can be defined specifically for the parent (master-detail)
2. The related items can be selected from a predefined list

Many-to-one

In many-to-one relationships the parent item is referring to one child item, while the child item can get referred to by many different parents. An example at TEJ is a voyage that is related to a vessel, however one vessel can be used in several voyages. For many-to-one relationships the user usually is required to make a selection from a list of items. For example when defining a voyage he should select a vessel from a list of different vessels. There can be situations where the selection is optional, and situations where the selection is required. Another distinction can be made between static lists (for example a list with client-types, that only contains a shipper and a receiver) and lists where the user is allowed to add values to the list (for example in a list of clients).

Many-to-one relationships have the following properties:

- One item is referring to another item (often from another entity)
- The relation can be optional or required
- The list of items can be static or dynamic

5.3 Access control

At TEJ, but I think it is safe to say this applies to almost any company, they do not want anyone to have access to all their data. Therefore some kind of access control would be required. In the case of TEJ, where all employees are equal and no one else but the employees are allowed to access the system, it would be sufficient to have a simple login system that would provide access to the system. After providing a correct combination of a username and password the system should allow access to all data.

To be prepared for more advanced access control, for example where other actors are also allowed to access the system or when employees should have their own credentials in the future, the system can be prepared with a Role Based Access Control (RBAC) back-end. With such a back-end permissions can be assigned to users, by making users member of a group (where groups have permissions assigned to them) and by directly assigning permission to a particular user. An addition to this can be to add permissions per owner state, so that for example client-companies can only open their own assignments. This would require an extra layer of credential-checking that will also check data besides the structure, like in a Discretionary Access Control (DAC) system.

To protect the data in the system, the system can be equipped with:

- A login system
- A more sophisticated RBAC back-end
- An even more sophisticate system that not only checks the structure but also the content (for owner permissions just like in a DAC back-end)

6. Tools and Techniques

Development tools were searched to support the implementation of the described functionality in order to fulfil the requirements mentioned above. These tools should most of all support fast, simple and maintainable development, to allow making quick changes during the development iterations and to make the implementation of possible changed requirements in the future easily possible. The term that describes this process is called Rapid Application Development (RAD). It supports early development, with iterations in order to make the application more and more feature rich. *[Zaniotto & Potencier, 2007]* In this section I will describe some of the available tools and techniques that can accomplish this.

6.1 Database Management Systems (DBMS)

Before searching for a tool that would support RAD, it is important to see the big role Database Management Systems (DBMS) can have in simplifying the development process. Since databases contain and maintain all the data where these business support applications are all about, a good integration of a DBMS is therefore key in making lower development times possible.

Storing and retrieving data

R(elational)DBMS are the today's most widely used database systems. They store data in tables, similar to spreadsheet applications (see table 6.1): for every entity in an application a table is designed in the database, this table has a column for every property of the entity and every row (or record) contains the information of one item. Usually every item is also given a so called primary key, which is unique and makes it easier to distinguish the item. A specific query language (SQL) has been designed to perform operations on the data. Creating, retrieving, updating and deleting of records are the the most common of these operations, we will see these again in section 6.2.2.

Country Table		
country_id (*Primary Key*)	name (*Text*)	abbreviation (*Text*)
1	The Netherlands	NL
2	Belgium	BE
3	United Kingdom	UK

Table 6.1: A simple table with a Primary Key making a total of three columns

Maintaining relationships

RDBMS not only store items as rows in a table, they can also keep track of the relationships among the different tables and their rows. Primary keys play an important role in this: their value is used as a pointer in other tables in which they are called foreign keys. An example of this can be found in table 6.2 where the country table is referenced by the country_id column. The relation itself that describes that the foreign keys of the country_id column from the city-table correspond to the primary keys in the country_id column from the country-table is defined somewhere else in the RDBMS explicitly.

City Table		
city_id (*Primary Key*)	country_id (*Foreign Key*)	name (*Text*)
1	1	Amsterdam
2	1	Rotterdam
3	3	London

Table 6.2: A table containing a foreign key to another table

Maintains the integrity of the data

Relationship can also have constraints on them, to keep the data in a consistent state. A relationship can for example have a constraint that gets triggered when a primary-item of the relationship gets removed; in that case it can perform an action to also remove all foreign-items that refer to this item. In the example above that would mean that if you remove the country "The Netherlands", the cities "Amsterdam" and "Rotterdam" would also get removed from the database. Another possible action can be to prevent the deletion of the country, as long as there are cities related to it. Besides constraints, other triggers can be set in a RDBMS as well, to make the database perform actions automatically and keep the data consistent. In the case of TEJ an example of such a trigger is the updating of the old ETA, which gets set to the previous ETA as soon as a new ETA is added to the system.

Implementations

An overview of the more popular DMBS that can be used are :

- **Oracle** Innovator, powerful but expensive
- **Microsoft SQL Server** Runs only on Windows, integrates well in .Net environments
- **MySQL** Free, Open Source, popular, integrates in many environments, lots of documentation, fast, feature-rich
- **PostgreSQL** Free, Open source
- **SQLite** Simple, integrates in a program instead of running standalone, but functionality is limited

As can be seen there are quite a number of solutions. Besides SQLite, all DBMS offer similar functionality. Therefore it will be wise to first continue the search for appropriate development tools, after which a compatible DBMS will be selected.

RDBMS are there to:

- Store and retrieve data fast
- The data is structured
  - Every table matches with one entity from a data model
    - Every column matches with a property of an entity
    - Every row contains one item of the entity
- Keeps track of the relations among the data
  - With this knowledge it can maintain the integrity of the data

6.2 Techniques

In this section individual techniques are being described that can contribute to an improved maintainability of an application.

6.2.1 Object Relational Mapping (ORM)

Continuing with DBMS and their important role as just has been described, it is surprising to see the big difference between the database world and the programming world, making tight interaction difficult. Databases for example are structured relational, where development environments use objects. Another example is that both have different data types. Finally, getting access to this data is also done in a completely different way.

Without tools, a developer would need to define tables in a database and matching objects in his development environment, according to the designed data model. After that, an interface should be defined that connects the two, by converting communication during requests and results. To improve this situation, a development tool was searched that could integrate a DBMS into the development environment, making it possible to reduce the workload of development.

First of all since the tables in the database and the objects in the development environment are this much related (they are both implemented according to the same data model), it is not hard to recognise this can be exploited. Object Relational Mapping (ORM) is a technique designed to accomplish this. It maps the interface of a RDBMS to objects in an Object Oriented development environment (it converts the actions that are applied to an object to database queries, see figure 6.1), which makes it possible to transparently access and manipulate the data stored in the database directly from the objects. *[Zaniotto & Potencier, 2007][Wikipedia - ORM]* This reduces the repeated writing of basic Create, Retrieve, Update and Delete (CRUD) procedures for every object in the application and can so reduce the chance of errors and vulnerabilities, like for example well known SQL injections.

The use of ORM can introduce overhead, because complete objects will be filled with data, also if you only are interested in only one property of an item. This happens occasionally in overviews that show items and related properties. For example the ETA-list is an overview of voyages, for which (among others) related agents will be retrieved only to be able to show the name of the agent. However all other properties of an agent, from his address till his VAT-number, will be retrieved from the database and filled (hydrated) in the instance of the object. This overhead can be overcome by replacing heavy queries by custom code, but this has to be done with caution since it can introduce security issues and will influence the robustness of the application since changes in the data model can brake the custom code.

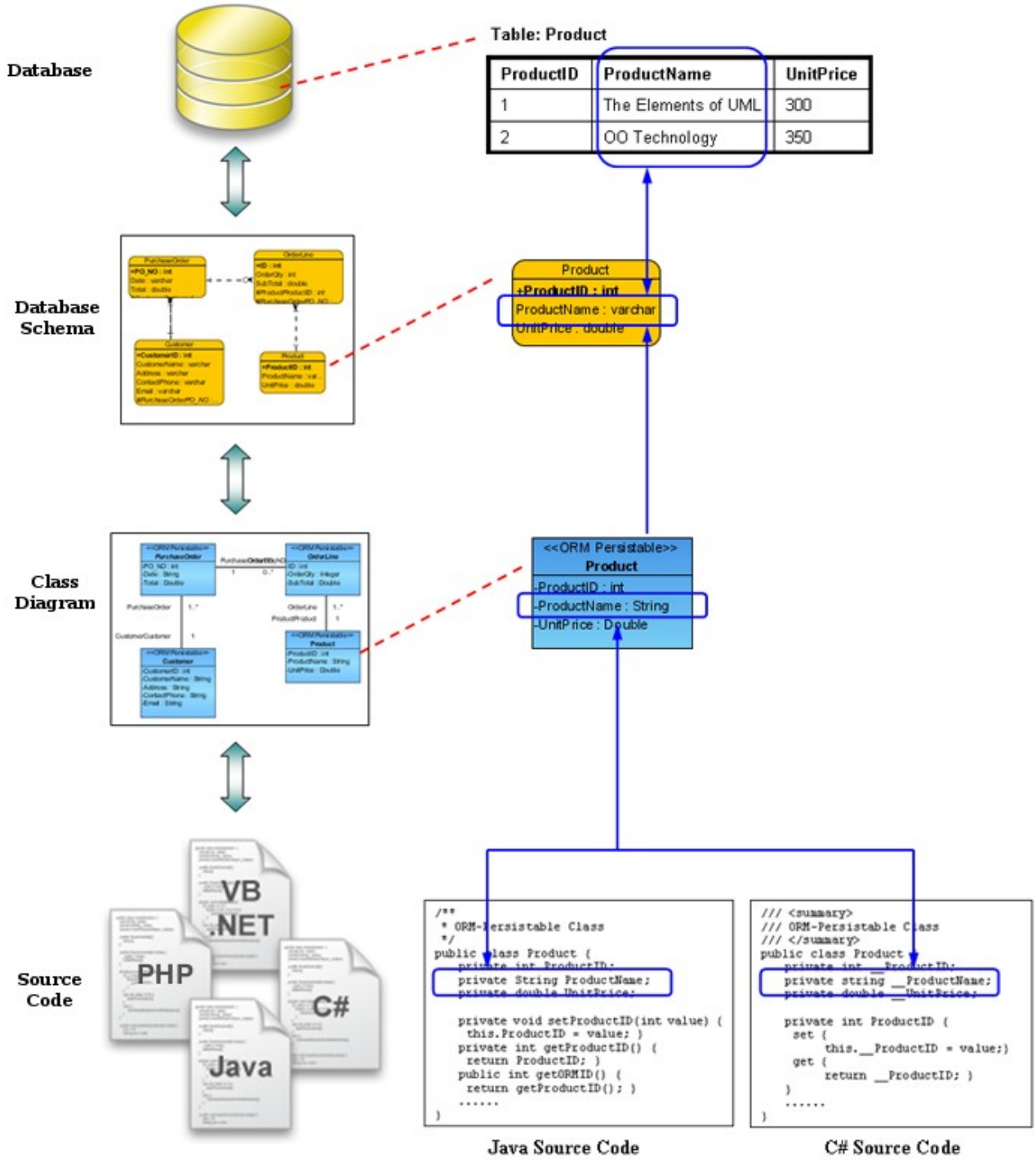


Figure 6.1: Overview of ORM: mapping the interface from a database to objects in a development environment

ORM is used to:

- Transparently access data stored in the database with objects in the development environment, this
  - \+ saves work since standard CRUD actions are automatically defined
  - \+ reduces change on errors and security risks
  - \- has as downside that it introduces overhead, since full objects get hydrated every time (but custom code can fix that in case efficiency is required)

6.2.2 Scaffolding and Administration Generation

Just like the tables in the database and the objects in the development environment that are both related to the data model, the user interface of business applications is often strongly related to the data model as well. The interface should allow the user to see and manipulate the data contained by the application. This pattern has again been recognised already and techniques have been designed according to the DRY-principal (Don't Repeat Yourself) to exploit this: further reducing the workload for the developer.

Scaffolding

Scaffolding is a technique designed to overcome the repeated implementing of basic actions and views for the manipulation of entities from the data model. It is used to be able to quickly generate a basic structure around a data entity, that supports the standard Create, Retrieve, Update and Delete actions (CRUD, see section 6.1). The code generated is intended only for testing and prototyping, but can serve as a base that should be altered and extended to build a more powerful application upon.

The big problem with scaffolding is that whenever the data model gets changed, the generated code from the scaffolding will not get updated accordingly. You can of course perform a new scaffold action, but this would replace all customisation done on the previously scaffolded code. Another solution would be to update the customised code further to make it reflect the current data model. This would require you to write the basic actions and templates manually for the changed fields, something which was possible to avoid because of scaffolding.

Scaffolding is used to:

- Create basic CRUD-views for an entity based on the data model
  - o These views are intended as base to build a more specific solution with
  - \+ Reduces development time
  - \- Not robust, incapable of quickly adapting to changes in the data model

Administration generation

Fortunately also a more flexible solution similar to scaffolding exists that is much better capable of handling any changes made to the data model. This technique is called administration generation (also known as dynamic scaffolding) and combines the information from the data model together with a configuration per data item (the modules that can be seen in figure 6.2), with which you can define all kind of properties about the actions and views that are being generated to view and/or manipulate the item in a user interface component. [Zaniotto & Potencier, 2007][US Patent 7240070] The configuration of the administration generator defines a higher level of abstraction above a development language, with which you can reduce a lot of the repetitive tasks that were else needed during development. Patterns that are recognised, like the need for overviews and detailed views on items, can be exploited by defining templates that are transformed into code by the generator. This will reduce the development time and the lines of code (loc) required to define views, that way improving maintainability compared to plain scaffolding, let alone programming in a regular development language, while reducing the possibility on making errors.

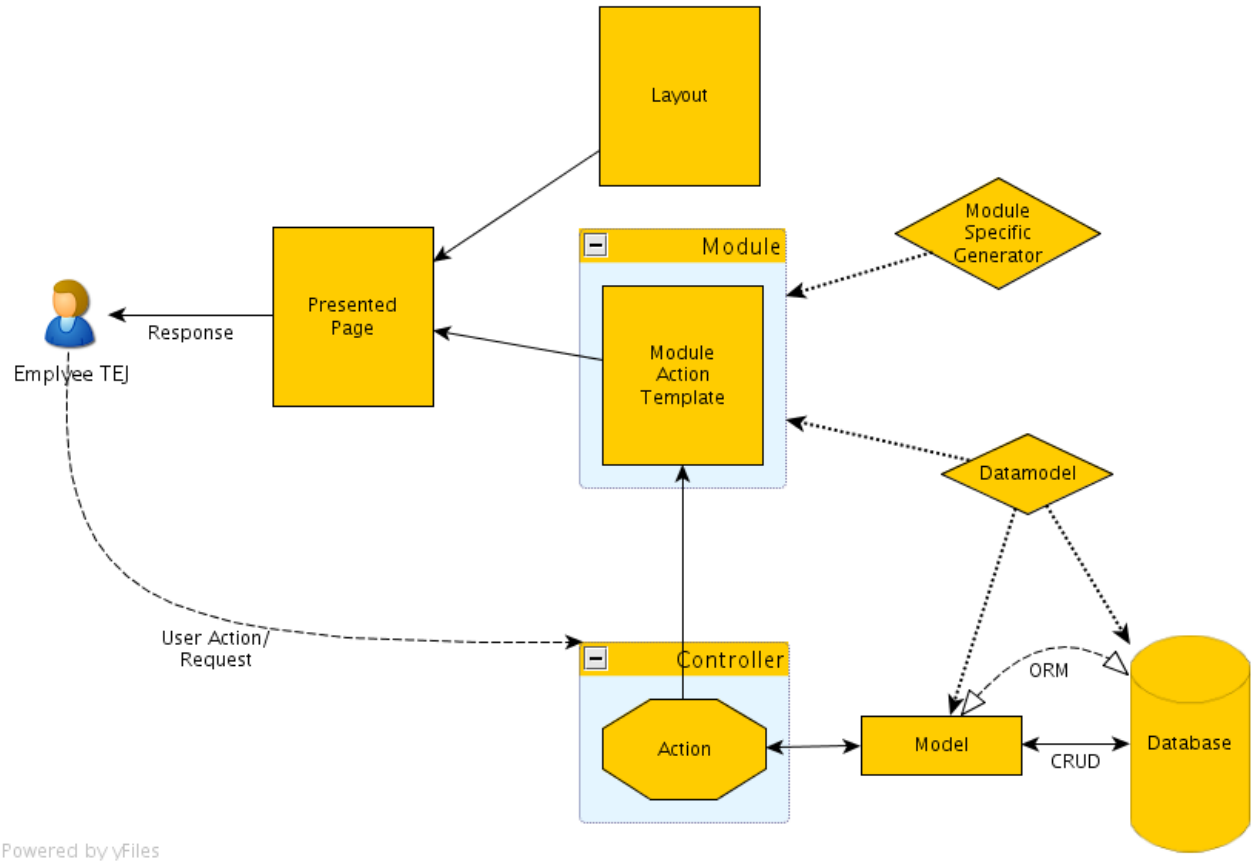


Figure 6.2: Administration Generation as implemented in the Symfony Framework

Administration generation cannot do magic. Its possibilities are limited to predefined functionality (defined in templates, combined with information from the data model and a configuration). In the case described here that means showing an overview of items, or allowing the manipulation of a specific item according to the data model. If the possibilities of the generator are too limited, the resulting code after generation can be extended, enhanced and replaced with custom code that allows you to make the result look and behave just like intended. Even though this makes administration generation extremely flexible, you should be aware that replacing generated code or even setting many properties in the configuration should be used with care since this introduces the same problem as with scaffolding: it makes it harder to keep the application match the data model in case it gets changed. Fortunately the changes that are required to make the actions and view match a new data model are much smaller and so still faster and easier to implement.

Administration generation is used to:

- Create views for an entity based on the data model and a configuration
  - Custom code can extend, enhance and replace generated code
    - Be aware that custom code is harder to maintain, just like with “normal” code
  - Patterns can be exploited by templates for the generator
  + Reduces development time and change on errors
  + Relatively easy to adapt to changes in the data model
  - Writing templates takes some time getting used to

6.2.3 Model-View-Controller (MVC)

Another nice technique developed to improve the development process is the Model-View-Controller pattern (MVC). This pattern defines a way to decouple the data and business logic from its presentation. A controller layer is used in between the user interface and the model to allow the user to interact with the data (see figure 6.3). This pattern makes it easier to reuse code according to the DRY-principal, again reducing complexity and improving maintainability.

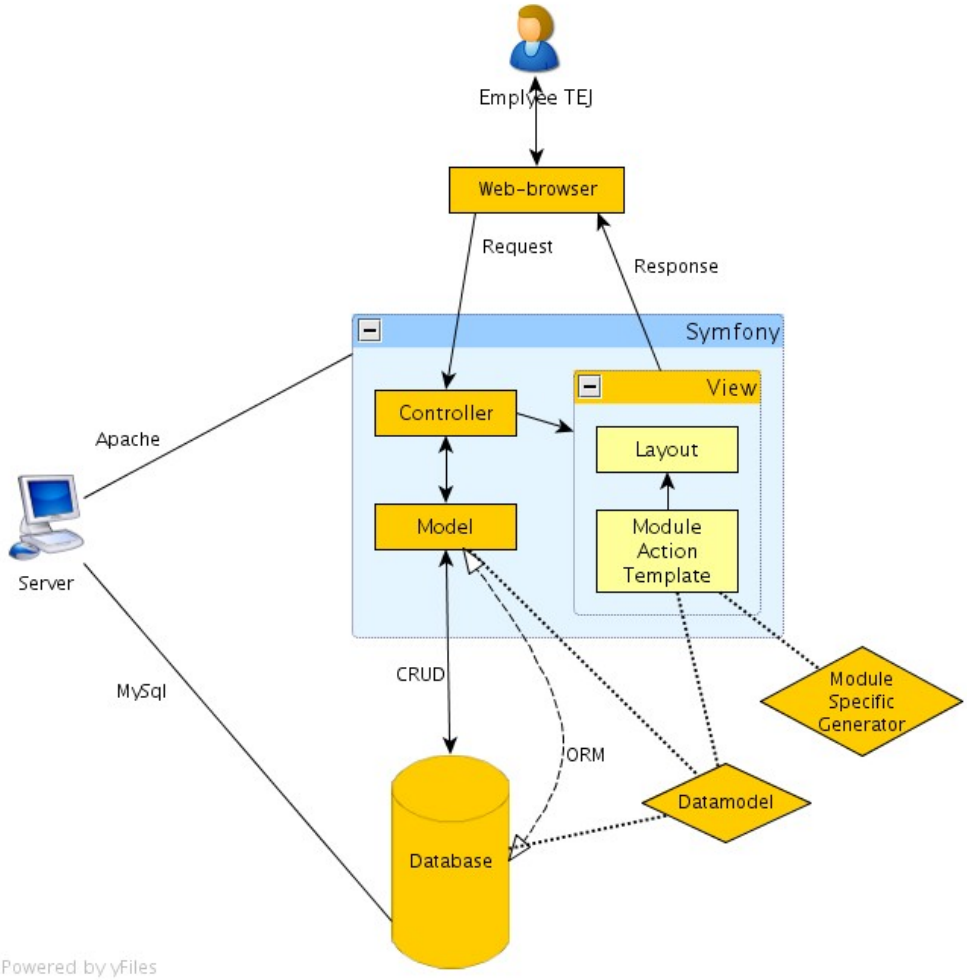


Figure 6.3: An overview of how the MVC pattern is applied in the Symfony framework

The MVC pattern is used to:

- Separate the business logic from its representation, by using three categories:
  - The model *to represent the information in the application*
  - The view *to show the data*
  - The controller *to allow the user to interact via the view with the data*
- Improve the development process by:
  - Improving the maintainability of code
  - supporting the reuse of code

6.2.4 YAML

Another technique that has been used during this project and which is capable of improving development is YAML, which stands for “YAML Ain't a Markup Language”. YAML is used extensively within Symfony (the framework I have chosen to use and which will be described in section 6.4) and is a standard that defines a way to describe structured data, just like the more well known XML-standard. However instead of using tags and brackets to define the structure within the data it uses whitespacing, which makes it much better suited for human reading and writing. Symfony uses YML-files to allow the configuration of it and since these configurations are written by humans, YML is a real nice solution to improve the development process.

Below a small example is provided where some data is represented in both XML and in YAML to show the difference between the two. The example is taken from IBM's website[4].

In XML:

```
<club>
  <players>
    <player id="kramnik"
            name="Vladimir Kramnik"
            rating="2700"
            status="GM" />
    <player id="fritz"
            name="Deep Fritz"
            rating="2700"
            status="Computer" />
    <player id="mertz"
            name="David Mertz"
            rating="1400"
            status="Amateur" />
  </players>
  <matches>
    <match>
        <Date>2002-10-04</Date>
        <White refid="fritz" />
        <Black refid="kramnik" />
        <Result>Draw</Result>
    </match>
    <match>
        <Date>2002-10-06</Date>
        <White refid="kramnik" />
        <Black refid="fritz" />
        <Result>White</Result>
    </match>
  </matches>
</club>
```

Figure 6.4: Data serialised in XML

4 http://www.ibm.com/developerworks/xml/library/x-matters23.html

In YAML this same data of a club would be represented like this:

```
players:
  Vladimir Kramnik: &kramnik
    rating: 2700
    status: GM
  Deep Fritz: &fritz
    rating: 2700
    status: Computer
  David Mertz: &mertz
    rating: 1400
    status: Amateur

matches:
  -
    Date: 2002-10-04
    White: *fritz
    Black: *kramnik
    Result: Draw
  -
    Date: 2002-10-06
    White: *kramnik
    Black: *fritz
    Result: White
```

Figure 6.5: Data serialised in YAML

In YAML the same structures can be described as in XML or JSON (another syntax to encapsulate data, which will be described in section 6.5.4) in a very human readable way. Short hand notations are also available, for example to define arrays or hashes: You can comma-separate items and surround them with [] or {} respectively, which makes it a perfect language to describe structured data from associative arrays.

YAML is a very nice language for defining data structures:

- It is easy to read and write for humans
- Can easily be transformed to other languages like JSON and XML

6.2.5 Asynchronous JavaScript and XML (AJAX)

One last term not to be forgotten in this section is AJAX. AJAX is short for Asynchronous JavaScript and XML, it is a combination of existing techniques not directly to improve the development process, but to make a webpage exchange data with a server, without reloading the entire page. [Wikipedia - AJAX] This not only makes web applications respond faster, since only small amounts of data need to be transferred, but also more interactive since small parts of the interface can now be updated. This will improve the usability of the application, which is desired since the frameworks I have found (see section 6.3) were all not returning rich interactive interfaces.

By default the different frameworks generate plain HTML pages. They are dynamic in the sense that with every reload of a page they are filled with up-to-date relevant content, but they lack any form of dynamic interaction. This can be very useful for checking if data exists in the system, while you want to use it. For example when TEJ wants to add a new assignment for a client, but this client is not yet known in the system, the system can check this while the assignment is being added and

inform the user, so it can immediately be added. This is of course also usable for other situations like when adding a parcel while its product is not yet known by the system, but it also gives the opportunity to allow instant editing of values from for example overviews, without loading specialised detailed edit-pages, to make quick editing possible.

With the implementation of AJAX the separation of data and layout is also improved. In plain HTML, both the layout and the data are combined at the server in one page. This gets transported to the client, where it directly gets presented. With AJAX the client-side is provided with additional knowledge, to make it possible to combine the data and layout over there. With this knowledge actions can be performed at the client-side. It for example allows for (asynchronously) requesting data from the server, after a page has already been loaded, when the client application gets noticed the server data has been changed, or when the user request to refresh the data.

This all sounds nice, but there is a catch: there are several web browsers available and all of them seem to interpreted JavaScript and styling a little differently. Fortunately there is almost always a way to define something in such a way that it works on all browsers, and in the other situations different workarounds can be defined for every browser to make it work in those exceptional situations as well. This however requires a lot of knowledge about how the “standard” is being interpreted by the different browsers and therefore frameworks again can offer a solution. More about that can be found in section 6.3.3.

AJAX is the name for a combination of existing techniques:

- It combines JavaScript and XML to make websites more dynamic
- Allowing for more interactive interfaces
- Speeding up the interaction
- Several frameworks exists to provide solid solutions for browser incompatibilities

6.3 Framework selection

After some tools and techniques have been identified that make it possible to improve the development process, a framework was searched in which these are used or can be integrated. This means the framework should allow for OO-development, according to the DRY-principals and the MVC and RAD-patterns, using a flexible form of administration generation to make development faster and more maintainable.

6.3.1 Native application frameworks

When thinking about business software, the first thoughts that immediately went through my mind were about native (Windows) applications, since these are the kind of applications that are seen most often (and probably for a reason). However, at the time I was searching for tools that could be used to improve implementation, none provided me the techniques as described above. Microsoft offers a very nice environment for developing applications, called .Net. It is used often, but only at the time of writing this thesis Microsoft has made a tool available called LINQ. It bridges the gap between a RDBMS (and other data sources) and a development environment. It contains features like deferred execution of sub-methods that build up to a query with which data from an arbitrary data source can be manipulated. This sounds very nice, but as said it unfortunately was not available at the time I started developing.

6.3.2 Web server frameworks

Continuing the search for an appropriate development environment, that would offer the intended tools to improve development, brought me to the world of web development. This world is filled with custom made applications and websites, many of which are connected to database systems. Besides, many new technologies are being used in this world. This made it a perfect place to look for development tools that could deliver an improved integration with a DBMS and providing flexible tools to generate user interface elements upon a data model.

An extra benefit of developing a web-application is that the result is much more flexible: it does not require difficult installation procedures and it makes the system available from anywhere, as long as there is an internet connection available [Ziouvelou,Sideris & Nancy Pouloudi, 2004]. This way also reducing implementation and maintenance problems and costs [Markus, Petrie & Axline, 2001].

With the capabilities of Asynchronous JavaScript And XML (better known as AJAX) it is possible to create applications that feel like native ones. New browsers like the upcoming Mozilla Firefox 3.1 and Google Chrome aid to this by improving the runtime speed of JavaScript code by compiling it.

Several frameworks that contained the techniques described above were quickly been found. The two frameworks that turned out to be most promising were Ruby-on-Rails (RoR) and Symfony. Ruby is a relative new development language allowing for faster development and the RoR-framework provides some nice tools to improve this even further. Nevertheless the Symfony framework was selected, since it contained better documentation and provided tools that suited my needs equally well.

Before explaining how Symfony works, first a short overview of some of the (open source) web frameworks that are currently available will be given. This overview is not a real comparison, but gives an idea about the frameworks that are available and the arguments I had in favour for Symfony:

- **Ruby on Rails** *(Ruby)* - http://rubyonrails.org/
  - \+ New language, developed to reduce development time
  - \+ Active Community
  - ○ Enough documentation, but scattered
  - \- Not supported at many hosting companies
- **Symfony** *(PHP)* - http://www.symfony-project.com/
  - \+ PHP language well supported at many systems and hosting companies
  - \+ I am already familiar with the language
  - \+ Good documentation
  - \+ Active community
  - \+ Used at large corporations (Yahoo) and popular websites (delicious.com)
  - \+ Friends who already used it and have knowledge to support me
- **Django** *(Python)* - http://www.djangoproject.com/
  - \+ Supposed to be fast (run time)
  - \- Stable version released only just before I started writing this thesis
  - \- Not supported at many hosting companies

Symfony was selected relatively quickly, simply because it fulfilled most of the requirements. It

- \+ contained the tools I was looking for
- \+ is fully adjustable
- \+ has good documentation and information (from friends and an active community) available
- \+ has a stable release

The only drawbacks I could find were that the generated interface was

- \- not interactive enough (the reason to look for a different front-end, see section 6.5)
- \- too basic, for example only layout with fieldsets were available to group fields in a form

However none of the other frameworks would be able solve these issues out of the box either and since Symfony is fully pluggable I was able to customise things to my liking in order to solve these issues, if not yet done by others already.

6.3.3 Client-side JavaScript Frameworks

To solve the issue of not being interactive enough, web-applications can be enhanced with JavaScript for the browser at the client-side that contain an understanding of the business logic. As already mentioned in section 6.2.5, the big problem with JavaScript (and web applications in general) is that they can run in different web browsers at the client-side and each of these browsers have the tendency to interpret JavaScript and styling a little bit differently. Fortunately browsers are interpreting the standards more and more strictly, but still minor changes can result in annoying and even unusable situations.

However solutions to overcome this problem and to be able to offer backward compatibility with older browsers are available. It requires substantial knowledge about JavaScript, styling and the different browser interpretations, but then it is possible to handle the exceptions for all the different interpretations. Fortunately plenty of JavaScript frameworks exist that already contain this knowledge in one easy to use library and can so result in good working JavaScript and styling for all popular browsers. These libraries contain a combination of many common solutions, however for the office applications I have in mind only one stood out: Ext JS. I found it thanks to the Symfony community and up till now I have found only one possible alternative, although this one is yet again not stable at the moment of writing this thesis: Qooxdoo, a free open source Object Oriented JavaScript framework offering tools to create web applications, without requiring the developer to have any knowledge about HTML or styling [Qooxdoo website]. Ext JS will be further described in section 6.5.

Eventually only two frameworks were found suitable for creating office applications:

- Ext JS - http://www.extjs.com/
  - \+ Many advanced “office” widgets
  - \+ Contains objects for a communication layer
  - \+ Stable
  - \+ Based on best practices of other frameworks
  - \+ Good documentation
  - \+ Amazingly active community
  - \+ Extendible

- Qooxdoo - http://qooxdoo.org/
  - \+ “Office” widgets
  - \+ Contains objects for a communication layer
  - \+ Well documented
  - \+ Extendible
  - \- Not yet stable
  - \- Less support
  - ○ Only found after implementation in Ext JS has started

6.4 Back-end framework: Symfony

As described in the previous two sections, during the search for a potential framework to support rapid application development (RAD) it turned out it would be desired to use an additional framework to improve the interactivity of the user interface. The framework that would be used for RAD will run at the back-end. This is the part of an application where the actions and data are being processed. As explained, I have chosen to use Symfony for that and in this section I will go deeper into the working and possibilities of this framework.

6.4.1 Symfony

Symfony is a free, open source, fully pluggable, server-sided PHP-framework that supports object oriented agile (web)development. This is quite a mouth full and still does not tell you everything. It comes down to that it uses best practices available for the techniques mentioned in the previous sections, like the Model-View-Control (MVC) pattern to separate the knowledge and the logic of its representation. A fully Object Oriented (OO) environment is provided in which Object Relational Mapping (ORM) is used to transparently convert data between (PHP-)objects and the tables in a relational database. Finally scaffolding and administration generation tools are available to automatically create actions and views based on the model. The techniques are often found in other frameworks and are combined in Symfony as a set of cohesive but decoupled classes, which makes it very flexible; allowing you to replace and extend anything you desire. [Symfony website] The administration generator can, just like the rest of the framework, be configured, altered and extended (with the help of plugins) to make it do whatever is desired. This all helps to reduce the development time and makes it possible to deliver a user centred product.

6.4.2 Implementation of ORM

Definitely one of the most important features integrated in Symfony is the ORM. In Symfony there are currently two different ORMs available: Propel and Doctrine. Both ORMs are available for PHP and where Propel is currently integrated by default in Symfony, Doctrine can be installed as a plugin to replace Propel completely. I have chosen to continue using Propel, since Doctrine was (again) not stable at the time I started development. Doctrine is much better capable of handling relationships and has just been released as a stable while writing this thesis. Changing it in the future is possible and would only require rewriting some custom code that is added to refine the model classes, which will make the application more maintainable.

Propel has several tasks within Symfony. As could be expected from an ORM it offers the possibility to transparently access and manipulate the data stored in the database from the objects in the development environment. In order to be able to do this it generates base-classes based on the tables defined in the database. So called peer-classes are constructed, which contain information about how to retrieve the items from the database and put them in the constructed objects. Finally wrapper-classes are provided that extend these base-object- and base-peer-classes, to offer the possibility to enhance them with custom functionality (more about this in the section below).

Besides generating these classes, Propel is also capable of constructing the structure for the database itself, based on a description of the datamodel. By default this description should be written in XML, but Symfony provided an extra reader for this that is capable of converting YAML to XML,

which makes defining this description real easy (see figure 6.6). Another way to construct the classes is possible by providing Propel an already existing database so it can base the model definition directly from the database. [Propel website]

```
propel:

  # countries
  country:
    country_id:
      type: INTEGER
      required: true
      autoIncrement: true
      primaryKey: true
    name:
      type: VARCHAR
      size: 150
      required: true
    abbreviation:
      type: VARCHAR(4)
      required: false


  # cities
  city:
    city_id:
      type: INTEGER
      required: true
      autoIncrement: true
      primaryKey: true
    country_id:
      type: INTEGER
      required: true
      index: true
      foreignTable: country
      foreignReference: country_id
      onDelete: RESTRICT    # you cannot delete a country as long as
                              there is a city related to it
      onUpdate: CASCADE
    name:
      type: VARCHAR(150)
      required: true
```

Figure 6.6: Example of the definition of the data model in a schema.yml file

Properties of the ORM within Symfony:

- Capable of setting up the database, creating tables with columns, based on a description of the data model
- Capable of generating the object model, with peer classes to retrieve multiple items, based on the data model or an existing database
- Allows for customising the objects from the object model, so you can implement all business logic
- Allows for customising the peer objects accompanying the object model, to be able to define advanced relationships and for performance tuning
- Plugable, the default ORM Propel can be replaced by others like Doctrine

6.4.3 Object Model

As already quickly mentioned above, the ORM (Propel) is used to create the object model and the accompanying peer classes that are capable of retrieving items from the database. The structure of the base and extended (peer)classes is arranged as followed:

- model
  - base Object Model folder
    - BaseClass *contains the generated class of an item*
    - BasePeerClass *contains the generated peer class of an item, to retrieve items*
  - ExtendibleClass *contains the custom code to extend the base class of an item*
  - ExtendiblePeerClass *contains the custom code to extend the peer class of an item*

The "base Object Model folder" contains the generated classes from Propel, while the corresponding extendible sub classes are being maintained in a folder directly below. This way the object model can be regenerated by propel when a new data model has been defined, without destroying all custom code written in the extended sub-classes.

The base classes contain all properties of an item, together with getters and setters to manipulate the data in an instance and actions like hydrate, delete and save methods to interact with the data from the matching row in the database. The extendible class can be used to add custom methods to the classes, for example to define "toString" methods that return a representation of the instance in text. In case of for example country objects this can be the country name, or a combination of the country name and the country abbreviation like: "the Netherlands (NL)".

The Peer classes contain information about the related table in the database. It is used to find and retrieve rows from the database with which related objects will be hydrated. The extendible peer class can be used to define custom peer methods. This makes it possible to retrieve items together with related items, which not only speeds up the ORM-process but also makes it possible to define the type of relation. Cities are for example related to countries with a (required) many-to-one relation: one city belongs to one country, but a country can contain multiple cities. A peer method for a city can be defined to not only retrieve a city, but also the related country in one query. For optional one-to-many relationships, like for example in the case of TEJ, where a voyage can contain 0, 1 or multiple assignments, a peer method can be defined that retrieves all assignments with their related voyage, but also the voyages that do not have any related assignments.

The objects in the object model:

- Contain the descriptions of the entities used within the application
- Are accompanied with peer methods to retrieve items from the database
- Are connected to the related tables in the database, to allow direct saving and deleting of objects from the database
- Are extendible with custom code

6.4.4 Applications

In Symfony the model that has been defined for a project can be shared by multiple applications (see figure 6.7). This way you can define for example both a back- and front-office application: One for the employees and the other one for clients. They can both have their own look-and-feel and capabilities, while working with the same data via the object model.

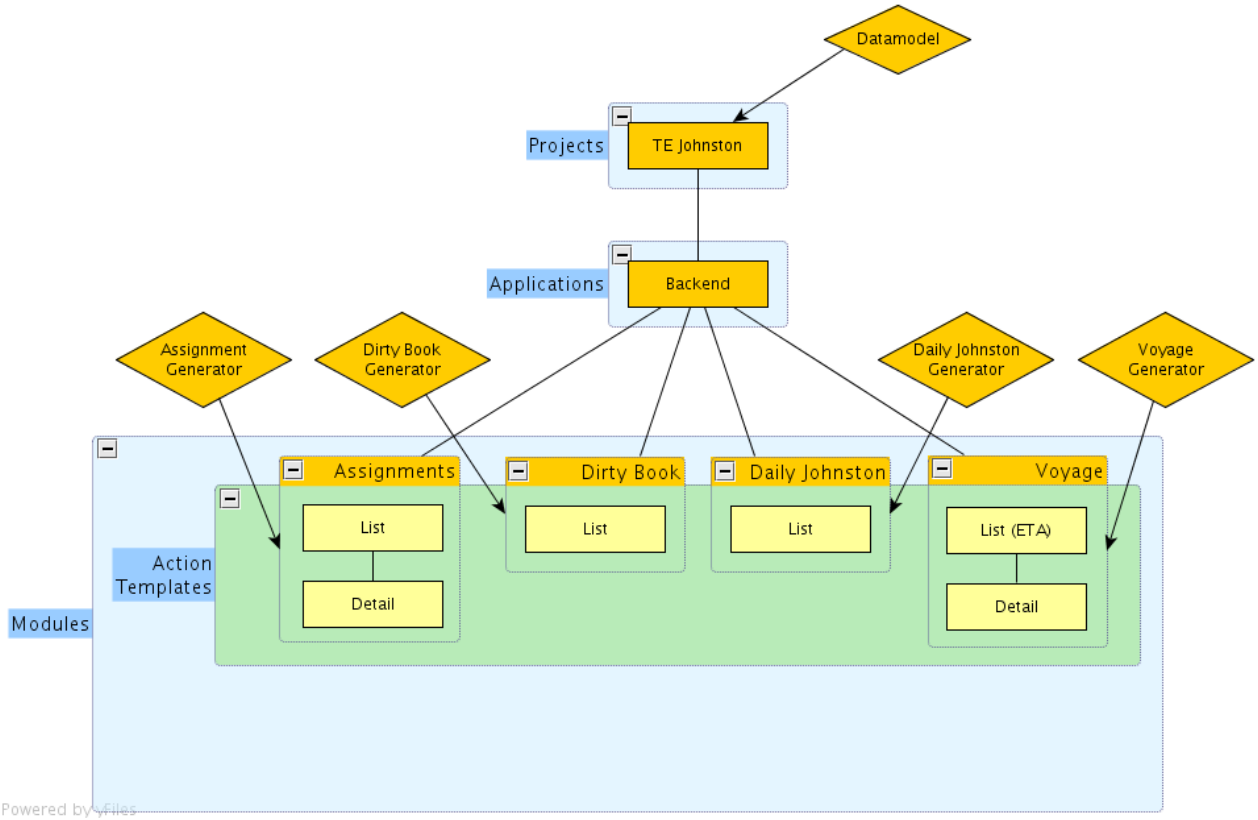


Figure 6.7: Project structure in Symfony

Applications are structured with the help of directories, just like the model library. A simplified overview of this structure is given below:

- application
  - config
    - app.yml
    - routing.yml
    - security.yml
    - settings.yml
  - modules
    - module*
      - actions
        - actions.class.php
      - config
        - generator.yml
      - templates
        - template.php*
  - templates
    - layout.php*

*items with a * can be zero, one or many*

As can be seen, each application has its own set of configuration files. These files make it possible to configure things like specific security or look-and-feel settings per application. The layout file in

the templates folder can further define this look and feel. However the most important part is the module folder, which contains the specific information related to the representation and handling around entities from the object model.

6.4.5 Generating Modules

Modules are used within Symfony to organise code for the user interface and actions, grouped per purpose. The MVC pattern (see section 6.2.3) is used to separate the controller code from the view-templates: an actions-folder contains an actions.php file to describe the actions for the controller, while a templates folder can contains several templates to define the view for every action (see the overview above in section 6.4.4). This code can be entirely written by hand like in any ordinary development environment, however generators can also be used to simplify this task.

Symfony can by default generate views and actions for entities defined in the data model, to show lists of items and detailed-views per item. The lists can be enhanced with filters to be able to narrow down the overview to items with specific properties. The edit-views are used to create new items with and to manipulate existing ones. Per entity from the data model zero, one or many modules can be defined to describe a user interface around them.

Example configuration

The configuration for this is defined in the generator.yml file in the config folder of each module. A simple example of this is given below:

```
generator:
  class:              sfPropelAdminGenerator
  param:
    model_class:      City
    theme:            default

    fields:
      country_name:
        name: Country

    list:
      display: [=name, _country_name]

    edit:
      title:   "Edit city"
      display: [name, _country_name]
```

Figure 6.8: Example of a generator configuration

With this generator.yml a module will be generated that can show lists and detailed edit-pages for items of the “city” entity. This way multiple modules can be defined for a same object from the data model. The fields which are defined in the display-arrays are defined in the data model of the model class (as described in figure 6.6). This lists will show the city name and country name, and so do the detailed edit-pages. In both views the title for the country_name-field will be labelled as “Country”, which is specified in the fields section. The actions for either view are not specified but are automatically created, based on predefined defaults of the theme.

Caching and customisability

The generated code will be stored in a cache folder of Symfony, which makes it unnecessary to generate PHP-code every time a page is being requested and so improves performance to regular levels. The code in the cache folder is split into several smaller template files that are being combined during execution. This way you can also use them for tweaking, by copying them from the cache into the modules template folder and adjust some code, which will overrule the matching template from the generator in the cache. This makes the generator extremely flexible (providing it with the same capabilities as ordinary development), but should be used with some caution since it will decrease maintainability.

The problem with the default capabilities of the generator

Unfortunately there are some problems with the generated code from the default theme. First of all the result is plain HTML, so not really interactive: Only feedback is given after an entire page has been posted and an entirely new page is being returned. Another problem with the default theme is that you are not able to define foreign values (like the related country name) directly from the generator. As can be seen in the generator.yml file, the "_country_name" field is preceded with an underscore. This is a convention within Symfony to define that this field is further defined in a partial, which is a small template with custom code to return a display value. This not only introduces additional code needed to be written, increasing the chance on errors and making the application less maintainable, but also makes the application lose knowledge about the relationship. This knowledge could for example be used for sorting a list on a related value, like country names, or make a field be able to auto complete. For this last issue Symfony has implemented a solution: By defining only the foreign-key Symfony will use the "toString" method of the foreign objects to allow showing related values. However a more powerful solution can be constructed, which will be defined in section 7.2.2.

The default generator will make it easy to create standard views for lists and detailed edit-pages per entity, everything generated can be tweaked, however some of the limitations with the default theme are that:

- The generated result will be plain HTML, so not interactive at all
- Foreign values (like the related country name for a city) have to be further defined in a "partial" with custom code, which makes knowledge disappear
- More configuration options are desired to reduce the need for overruling generated templates. For instance for a detailed edit-view only one title can currently be defined, regardless if the edit-view is for creating a new item (that does not have a distinguishable name) or if it is used to edit an existing item (where the name of the item you are editing would usually be a more appropriate title, like the city or country name).

6.5 Front-end framework: Ext JS

To improve the usability of the application, its user interface (which is the front-end of an application) should be interactive. Since the front-end of current web-frameworks was not interactive enough by default, there was searched for an extra framework that could be integrated with the back-end to accomplish this. As described in section 6.3.3, an AJAX framework has been found that contained many usable solutions for the office applications I had in mind, called Ext JS.

6.5.1 Ext JS

Ext JS is an open source client-sided JavaScript framework, which opens a way to create interactive web-applications. This library is based on the best practices seen in other frameworks, is very well documented, has an active community, is compatible with all major browsers and seems to be used by a lot of big companies [Ext JS Website]. It gives web-based applications the same look and feel as desktop applications (see figure 6.9). Together with the new browser developments (see section 10.4) it becomes harder and harder to notice the difference between web and desktop applications.

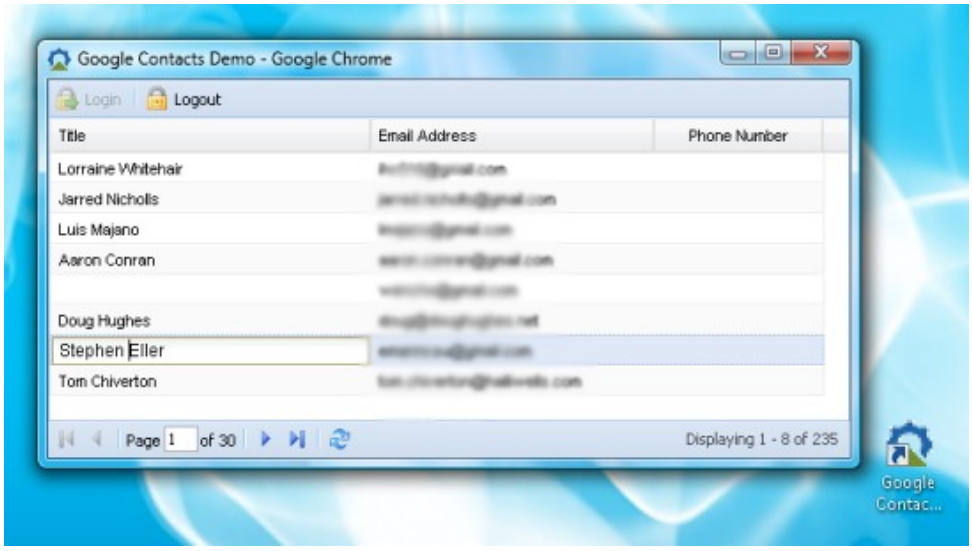


Figure 6.9: The gap between web- and desktop-applications is getting smaller

6.5.2 Grids

With this library it becomes possible to transform standard tables to interactive spreadsheets, like Microsoft Excel (one of the applications used extensively at TEJ). These views are called grids and are great for showing overviews on items, while allowing for quick and simple modification of the data.

6.5.3 Forms

It also makes it possible to structure detailed views on a specific item. Fields in a form allow to modify the properties of an item and can be divided over fieldsets, columns and tab-pages or anyway you like, without the need to reload data when a new tab-page is opened. It also allows for client-side validation to further assist the user with his tasks, just like it was already possible in desktop applications. Figure 6.10 shows a part of the detailed edit view for parcels that demonstrate this.

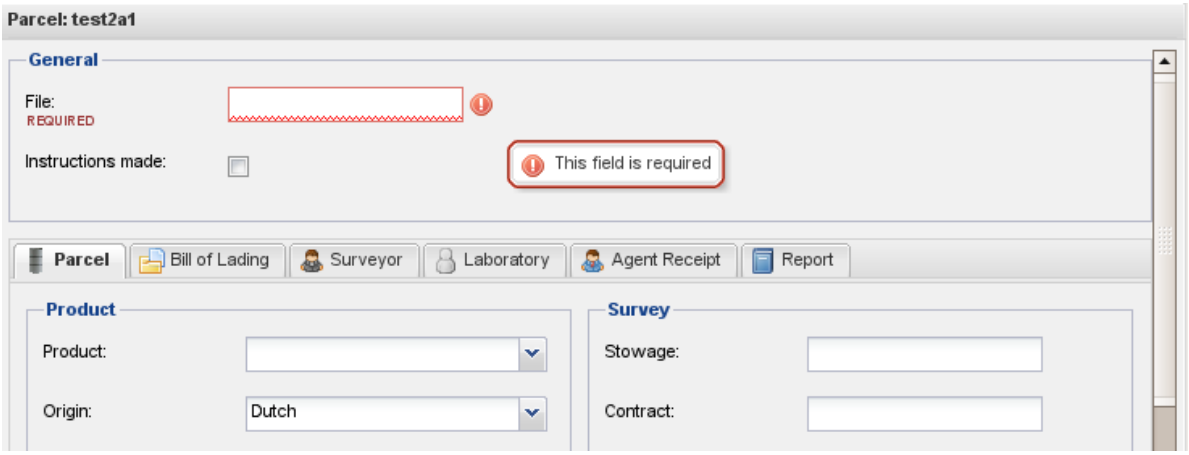


Figure 6.10: An example of a form with validation and where fields are divided over tab-pages, fieldsets and columns

Supporting many-to-one relationships

Relationships among entities defined in the data model can also be supported with the forms in Ext JS. Many-to-one relationships can be represented with so called combo-boxes as can be seen in the figure above for the product- and origin-fields. Both fields have the ability to complete the value the user is filling in, based on the related entities that are already known within the system.

Another way of how Ext JS is capable of supporting many-to-one relationships is by its ability to pop up windows. When a value is being entered in a combo-box that is not yet known within the system, it can pop up a window allowing the user to define and add this new item to the system, as can be seen in figure 6.11. After saving the new item, its value can directly be found in the combo-box where this entire process was initiated. In plain HTML you would always have to think in advance and first check for the existence of all related objects (like the product and the origin). If they do not yet exists in the system you should add them in advance, before you can define the new base object (like the parcel in this case).

In a similar way the user can immediately modify related items that are already known within the system. However this functionality is not yet available in Ext JS out of the box, but because of the flexibility of the framework it can be added with custom code. How this is done will be explained in section 7.2.2.

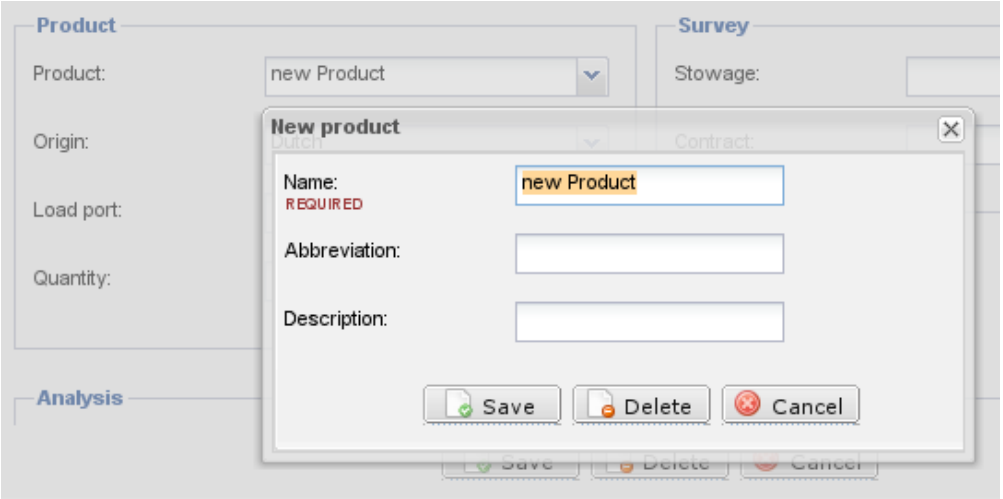


Figure 6.11: Pop up window to allow for instant adding of new related items

Supporting one-to-many relationships

Another solution that can be implemented relatively easy because of the flexibility of the framework is the support for one-to-many relationships. An example of this in the case of TEJ is the relationship of the Bills of Lading (B/Ls) to a parcel. Multiple B/Ls can be associated with one parcel and this can simply be visualised by a grid containing B/Ls shown in a detailed form for editing a parcel. That is where Ext JS again shows its power; because of its flexibility a grid would only require a small wrapper to make it appear as a field in a form (see figure 6.12). The data for this grid should be filtered by the server to only contain B/Ls that belong to the specific parcel, but this association can easily be set since the key to the parcel is already known in the form.

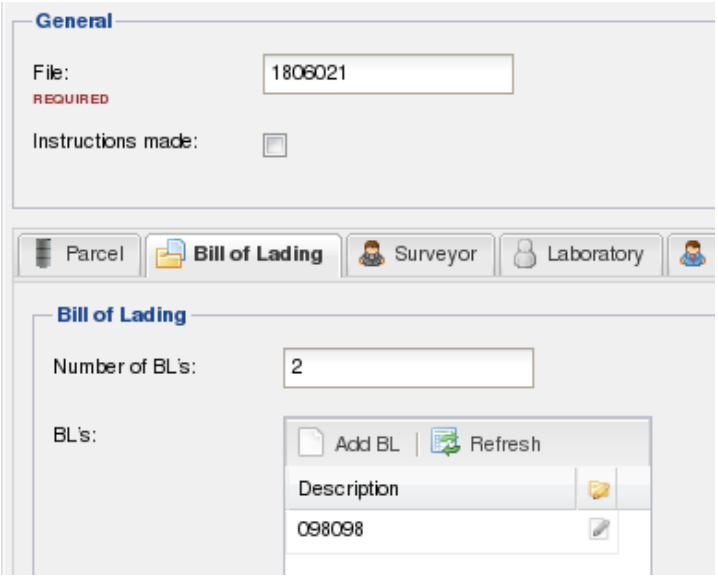


Figure 6.12: A parcel capable of containing mutiple Bills of Lading, shown as a list in a form

6.5.4 Customisability

The grid, form, fields and even the fieldsets and tab-pages are all objects defined in the Ext JS framework. Their constructor accepts a configuration from a config-object, which is an unordered set of (recursive) name/value pairs in JavaScript Object Notation (JSON). JSON is just like YAML a syntax much better readable by humans than XML and perfectly suited as data interchange format since it should be familiar to programmers of C-family languages (like C, C++, C#, but also Java JavaScript, Python and many others). [JSON website]

The configuration allows to set up objects in a simple way, however limited to the capabilities of the object. If you want to specify the behaviour of an object even further you can override and extend its default functionality with custom code. This is also useful to define preconfigured classes, which makes it easy to reuse code and improves maintainability. [Sakalos, 2008]

To support inter component communication, for example to reload a grid after some changes made in a detailed edit-page have just been saved, event handlers and listeners can be used; making development in JavaScript with Ext JS similar to development in for example Java or C#.

6.5.5 User Extensions

Ext JS has a very active community. They are very keen on answering questions (often answers are returned within an hour), but they also provide all kind of improvements and additions in the form of user extensions.

Three interesting extensions that are integrated in this product are:

- ext-basex - http://extjs.com/forum/showthread.php?t=21681

  A library with enhancements for Ext.lib.Ajax

  - *capable of monitoring XHR requests and providing events to handle exceptions (like login-requests or errors) to allow authentication and showing pop-up messages*
  - *support for synchronised requests (instead of asynchronous)*
  - *with an additional library (ux.ModuleManager) included that makes it possible to do lazy-loading (retrieving JavaScript code once it is required)*

- TinyMCE - http://extjs.com/forum/showthread.php?t=24787

  A wrapper around TinyMCE (a WYSIWYG text editor) to make it integrate with Ext JS

- IconMgr - http://extjs.com/forum/showthread.php?t=41281

  A utility class that dynamically adds styles for the famfamfam silk icons

Other extensions that are interesting are:

- LiveGrid - http://extjs.com/forum/showthread.php?t=17791

  The live grid can make the paging of results in the grids obsolete. It will fake the vertical scroll bar, making it appear all items are loaded in the grid, but in reality these are loaded on demand

- Print Preview - http://extjs.com/forum/showthread.php?t=35520

  At the moment you cannot print the grids, with the same make up as on screen. This extension should be able to fix that relatively easy

- Quick Filtering - http://extjs.com/forum/showthread.php?t=14503

  Adds the ability to filter on user defined criteria, directly from the column-menus in the grid

7. Sharpening the tools

In the previous chapter I have described some tools and techniques that could support the development process in making usable and adaptable business software. Frameworks were found that combined these techniques, but unfortunately none of them provided enough functionality to fulfil my requirements. To achieve the desired result the implementation of the administration generator had to be extended. A lot of work was required to be able to exploit the relations that are described in the data model and to improve the interactivity of the final result. This chapter will describe the process of how the Ext JS framework was integrated within the Symfony Framework and its administration generator was being enhanced at the same time.

7.1 Intermezzo: Open source

Before describing the extending of the Symfony framework and the merging with the Ext JS framework, I will first describe the role of open source in this project. Before I started the implementation I already knew that this project would be challenging. I am however a big fan of open source projects since I believe that a community of open source users are often found to be very enthusiastic and ready to help each other out. This provides ways to learn from each other, so in the end everyone gets better from it. The frameworks I had selected are both released as open source work, based on other open source projects in order to have best practices available. Both frameworks also have a large community of enthusiastic and active users, which would be a perfect base to get me started.

At the beginning of my project I made some simple customisations to the Symfony framework, to make the generator more powerful and to get me familiar with the Symfony framework. At that time another person (Wolfgang Kubens) had just started a thread[5] at the Symfony forum, which showed his ideas on how to improve the interactivity (by using Ext JS). I offered my help to collaborate with implementing his ideas and merge them with the work I had already implemented. From then on the implementation got a huge boost. Various people had offered their help during the implementation (see preface for some of the most active helping hands). The initial thread currently holds over 1100 messages and has been watched almost 150.000 times by people offering their ideas, improved the code and did testing while using the code themselves. As a bonus this kind of feedback provided an immense drive to improve the work. Besides this thread at the Symfony forum, several other threads have been started

Besides the forum, other tools like a wiki-page[6], a bug tracker[7] (Trac), a code management and repository tool[8] (Subversion) and chatting[9] (over IRC) were used to support the collaboration process. All code was implemented by defining a plugin in Symfony. This makes it possible to keep Symfony it self untouched and allowed everyone to check out a copy and work with it. At the moment several users and companies (among which the American Citibank) are using and improving the plugin, even though it has not yet been released as a stable version.

5 http://www.symfony-project.org/forum/index.php/t/5650/
6 http://raw.trac.symfony-project.org/wiki/sfExtjsThemePlugin
7 http://raw.trac.symfony-project.org/query?status=new&status=assigned&status=reopened&status=closed&component=sfExtjsThemePlugin&order=priority
8 http://svn.symfony-project.com/plugins/sfExtjsThemePlugin/
9 #sfExtjsThemePlugin at irc.freenode.net

7.2 The sfExtjsThemePlugin

The code that has been written to extend the capabilities of Symfony's default administration generator, and that integrates the Ext JS framework, has been combined in a plugin called the SfExtJsThemePlugin. The name starts with 'sf' by convention, like all Symfony plugins should and ends with 'Plugin' to denote the fact it is being a plugin (something which all extensions of Symfony can be since the framework is fully pluggable). This leaves 'ExtjsTheme' to be explained, although this probably is pretty self explanatory as well, since this plugin contains an (enhanced) theme for the administration generator that integrates Ext JS.

The functionality that has been added by me, with the help of the community, is more than meets the eye. Symfony could already generate views for overviews and detailed-edit forms, but as said in plain HTML, where the data is encapsulated by the layout. The possibilities provided by Ext JS make it possible to send the layout separately from the data. The layout that is generated for each object can be reused to show different items and therefore needs to be send to the client only once. This layout not only describes how to represent the data, but also contains information about how the data can be manipulated. For example to define how to subdivide fields in an edit-view over multiple tab-pages, but also to allow sorting on columns or validating input. Another difference with plain HTML is that not all data that is send to the client will be visible. The client therefore also gets equipped with datastores, that hold all data received from the server. (see figure 7.1)

With the sfExtjsThemePlugin the administration generator in Symfony does not return only one page of HTML-code, but:

- A (reusable) layout in JavaScript:
  - for the overviews of items, defined as:
    - a grid-object
    - a column-model (containing all columns with information on how to render its data)
    - a data-store (containing all fields with information about the data-types)
    - a data-reader (containing information about the connecting to the server)
  - For the detailed edit views, defined as:
    - a form-object (containing all fields, grouped with fieldsets and tab-pages)
    - a data-reader (containing information about the connecting to the server)
- Data, formatted in JSON (see section 6.5.4)

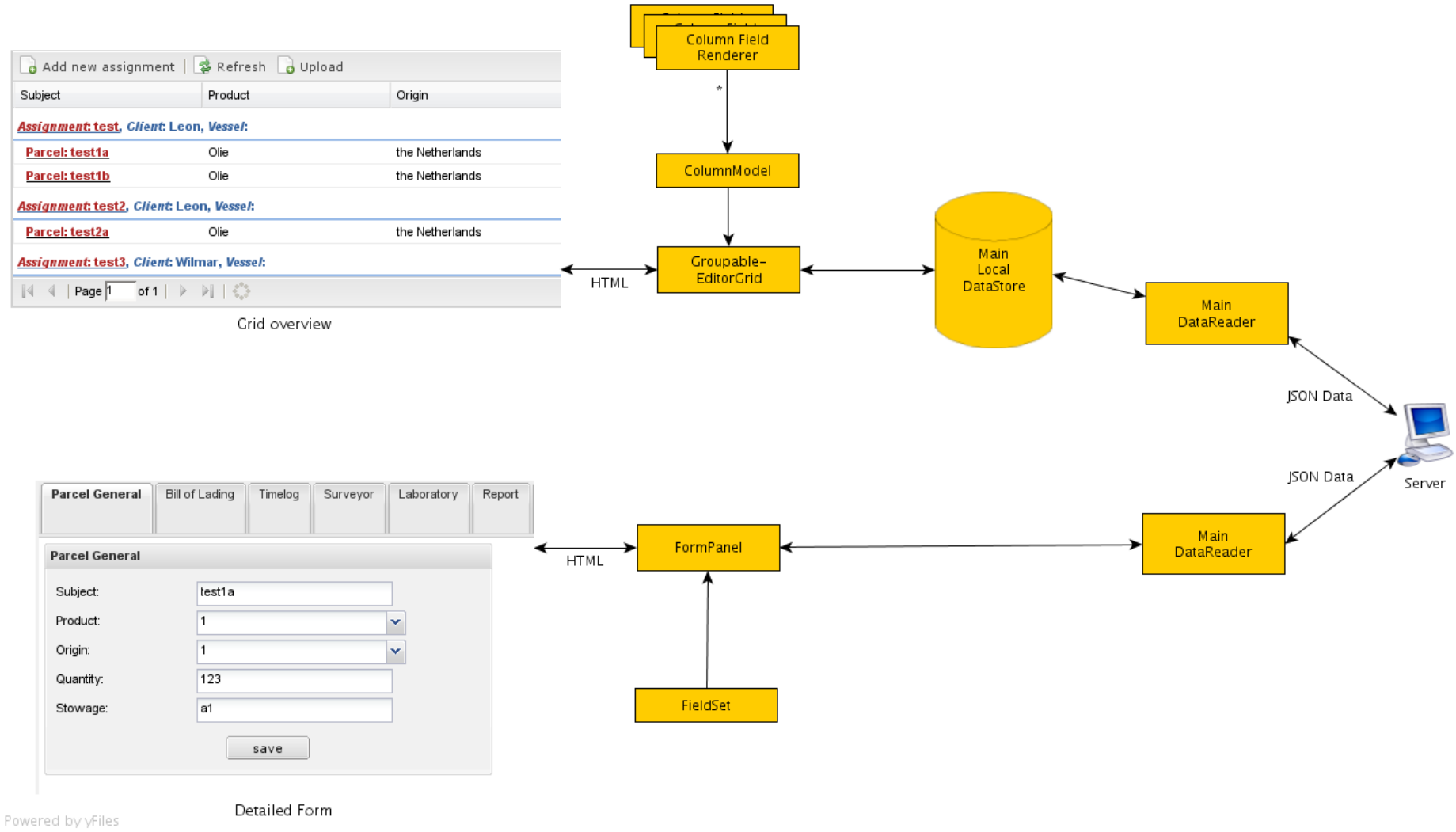


Figure 7.1: A simplified overview of how JSON-data is transfered and transformed to screen with the generated JavaScript objects

7.2.1 Data

The data contains different fields as required per view (overview and detailed as described by the corresponding configuration), per request (a specific page within an overview or a specific item in a detailed view) and per user (depending on the credentials). Chosen was to format the data in JSON instead of the more bloated XML-syntax, since it is easy to construct and parse by computers and very readable for humans, which makes debugging easier. See figure 7.2 for an example of a list of cities together with related countries, serialised with the JSON syntax, requested by a city-overview-grid. The totalCount Property is used to inform the grid about the total amount of results that are available. Since overviews can exist of thousands of records, you cannot always send them all at once to the client. The totalCount will in that case differ from the amount of items which are contained in the data.

```
{
  "totalCount": 2,
  "data" : [
    {
      "city_id"         : 1,
      "name"            : "Zwijndrecht",
      "country_id"      : 1,
      "country_id-name" : "the Netherlands"
    },{
      "city_id"         : 2,
      "name"            : "Rotterdam",
      "country_id"      : 1,
      "country_id-name" : "the Netherlands"
    }
  ]
}
```

Figure 7.2: An example of JSON data

7.2.2 Layouts

The generated layout consist of several objects (as described above) that communicate with each other in the user's browser. The objects are extensions of the default objects available from the Ext JS library (as described in [Sakalos, 2008]), preconfigured to the settings as defined in the module's configuration file (the generator.yml file like shown in figure 6.8, but updated with an extended syntax as can be found in the examples below) and based on the data model (as defined in the schema.yml file, like the one that can be found in figure 6.6).

Extended configurability

With the sfExtjsThemePlugin the configuration in the generator.yml file can be provided with more options than with the default administration generator of Symfony. This will improve the maintainability, since it prevents the need for custom code and keeps all configuration options together. Not all possibilities will be explained in this thesis, but some of the options that have been added to the syntax of the configuration file can be found in figure 7.3.

```
generator:
  class:              sfExtjsPropelAdminGenerator
  param:
    model_class:      AssignmentParcel
    theme:            extjs

    fields:
      subject:
        name:     File
        params:
          editable: false
          renderer: this.renderSubject

      bls:
        name: BL&#146;s
        params:
          xtype:       parcelblpanel
          key:         c.key
          parentPanel: this

    renderer:
      method:
        partials:     [_rendererSubject]

    list:
      title:       Assignment overview

      display:        [-assignment_id/subject, =subject]
      grouping:
        field:        assignment_id/subject


    edit:
      newtitle: "New parcel"
      title: "Parcel: <b>%%subject%%</b>"

      display:
        General: [subject]

      pages:
        parcel:
          title: Parcel

          display:
            "Product":    [product_id/name, quantity]
            "Survey":     [stowage, contract]
       bl:
          title: Bill of Lading
          params:
            layout: column
            iconCls: "Ext.ux.IconMgr.getIcon('folder_page')"

          display: [number_of_bls, _bls, bl_to_agent]
```

Figure 7.3: Sample of extended configuration possibilites available with the sfExtjsThemePlugin

Thanks to a community member (Benjamin Runnels), custom code defined in partials can be flexibly added to the generated objects from within the configuration (as for example a custom renderer method in the example above). These methods are written in JavaScript and the partials (with which Symfony was already familiar with) allow to organise them conveniently. This makes the generated objects completely customisable and the solution very flexible.

Another example where custom code is used often, besides the custom renderers, is in actions. Actions can be defined in the configuration, which show up as buttons in a toolbar. The code that is to be executed after pressing such an action-button can be defined directly as one of the properties when the code is small, but in other cases it can easily be defined in a partial that contains a more complicated method which will be added to the object.

Handle foreign values

Another difference between the default generator and the sfExtjsThemePlugin generator is that it now is possible to directly define foreign fields. This functionality is again based on the work of one of the community members of Symfony (Andre Schuurman), but improved to be able to handle multiple references to a same related object and applied to the Ext JS framework. An example of this new syntax can be found in figure 7.4.

```
generator:
  class:              sfExtjsPropelAdminGenerator
  param:
    model_class:      City
    theme:            extjs

    fields:
      country_id/name:
        name: Country

    list:
      display: [=name, country_id/name]

    edit:
      newtitle: "New city"
      title:    "Edit city <b>%%name%%</b>"

      display: [name, country_id/name]
```

Figure 7.4: Configuration for the generator of the sfExtjsThemePlugin

The syntax for handling foreign values is defined as foreign-key, separator, foreign-field-name (which can be recursive to allow for defining even deeper foreign values). This syntax not only makes it easier to define foreign fields, but also preserves the information from the data-model (something that would have been lost when using partials). This information can be used in the views to find the field on which you can sort the overview (see figure 7.5) and to assist in the form of auto-completion when you want to edit the value (see figure 7.6). In case you enter a value in the field that is not already known within the system (in this case for example an unknown country), the application can automatically detect this and pop-up a window to allow the user to add this item to the system.

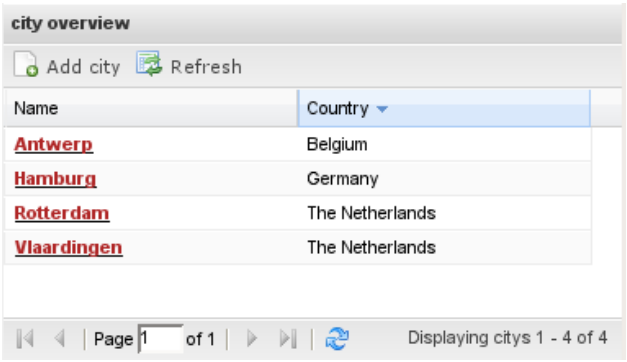


Figure 7.5: Sorting on a column with foreign-values

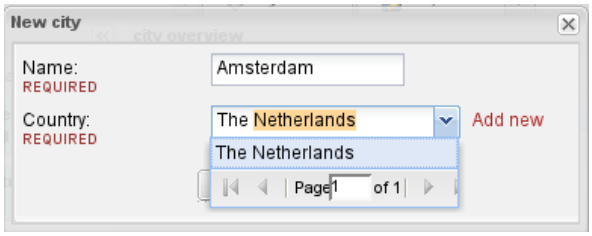


Figure 7.6: Autocompletion with alternatives after typing "The "

To make this possible, extra objects needed to be defined at the client-side, which connect to an extended interface at the server. Compared to figure 7.1 you will see additional datastores for every many-to-one relationship at the client-side, as can be seen in figure 7.7.

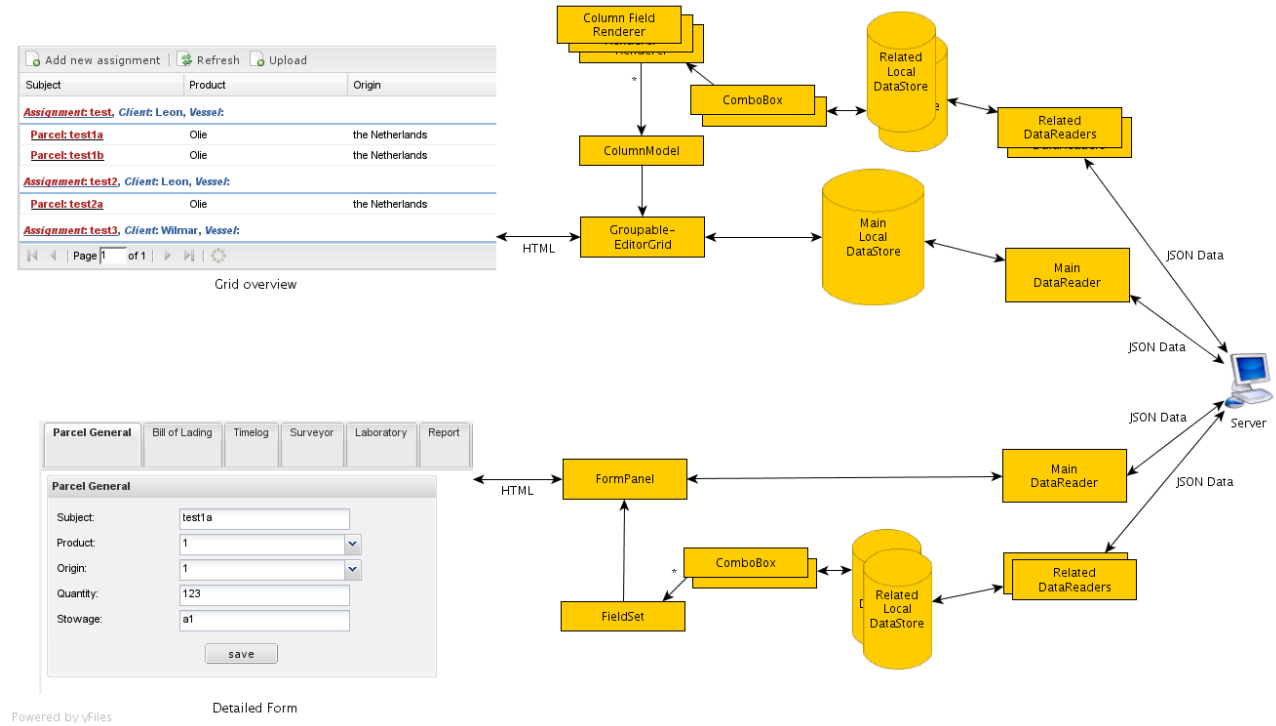


Figure 7.7: Additional DataStores are defined to support many-to-one relationships

Credential checking

Other functionality that is available to the generated fields and actions is for example checking for credentials to restrict the view for certain users. If a user does not have credentials to see certain fields these fields these will be removed from the view (as well as from the data) (again implemented by Benjamin Runnels). When a user needs to be logged-in into the system this can also be detected by the application so a pop-up window can show-up to allow the user to login (see figure 7.8). After a successful login the application will automatically repeat its latest request.

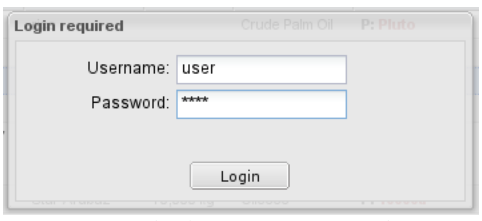


Figure 7.8: The login pop-up window

Validation

Finally, it also is possible to define validation rules for the data that can be entered. Currently client-side validation is limited to checking if all required fields have been filled (See figure 6.10) and if the entered data matches with the data-type that has been defined for a field. The check for required fields is done automatically by the plugin, with the information defined in the datamodel.

Extra validation can be added manually, however these are to be specified per module (which is at the controller layer), while validation of data should be done at model level. This can be improved by a separate plugin, but will probably also be implemented in a future version of Symfony and is therefore not yet been implemented in the current release of the plugin.

Improve Maintainability

The generator accepts many powerful configuration options to be able to generate overviews and detailed edit pages as designed, while reducing the complexity of development as already explained in the previous chapters. Since the generator returns normal code that would else had to be written by hand, it is easy to compare the lines of code (LOC) needed to configure the generator with the lines of code that are being generated, to get an impression of the simplification (see also figure 7.1).

For a simple overview plus detailed edit page on city-objects the **generator** would require **18 lines of code** (including empty lines and comments, as can be seen in figure 7.4). When counting the amount of lines that are being **generated** from this configuration, **44 files** had to be opened, adding up to a total of **2757 lines of code** that else had to be written by hand.

A more complicated module to generate for example both the detailed view on voyages and overviews for the ETAs will require some more lines for the generator: 97 lines to be exactly. However, also additional custom code is required: 54 lines divided over 3 files. This makes a total of **151 lines of code** divided over **4 files** for the **generator**. In this case the **generated** code was divided over **46 files** and exists of **3395 lines of code**.

	Generator		Generated Code		
	# files	*LOC*	*# files*	*LOC*	*Ratio LOC*
City Module	1	18	44	2757	153.17
Voyage Module	4	151	49	3449	22.84

Table 7.1: Comparison between Lines Of Code required to define a module

It is obviously that, because of these huge differences, the probability of making errors will be greatly reduced, while the maintainability of the code will be greatly improved (since only a few lines have to be checked and changed) when using the generator. Also configuring the generator will will require much less time compared to writing all this code. Not only because less code needs to be written, but all the difficult and technical implementation details are being hidden with the abstraction layer the generator defines. The reduced complexity is further endorsed when you know that in the generator only a few words are written on a line, compared to sometimes a couple of dozen in the generated code (see table 7.2).

	Generator		Generated Code		
	# files	*WC*	*# files*	*WC*	*Ratio WC*
City Module	1	27	44	6589	244.04
Voyage Module	4	310	49	8267	26.67

Table 7.2: Same comparison as in table 7.1, but now with Word Count (WC)

This improved maintainability has also proven itself in practice. Two big changes that were required, were the introduction of voyages (see section 4.1) and the relocating of invoices (see section 9.1). These changes affected all layers of the application, from the model to the view. Normally this would require the developer to modify the database, adapt the data-adapter (for reading and writing) to match the changes tables in the database, update the user interface and place the data from the updated data-adapter at the corresponding location in the user interface. This will affect the many lines of code in the application, as shown in the tables above. (Please note, these tables exclude the lines defined in the model-layer to define the data-adapters.)

With the use of the ORM- and administration generation-tools the only thing that is required is to update two lines in the schema of the ORM (from which the database is being created and a matching data-adapter is being generated) and one or two lines in the configuration of administration generator (to update the view that connects to the data-adapter). Custom code of course requires some more work, but should only be changed to match the generated data-adapter. One problem that currently remains is the migration of existing data in the new format. This is however currently being supported by Doctrine, the other the ORM available to Symfony.

7.2.3 Applications

An application made in Symfony with the help of the sfExtjsThemePlugin can (almost) completely exist out of JavaScript code that is send to the clientside (instead of HTML). As said, this allows for separation between the layout and the data and allows for a more interactive and responsive user interface. Layouts only have to be loaded once after which they can be reused for representing different items over and over again. This is completely different from how HTML is working, where every action requires a page to get completely reloaded. The base template that surrounds the content now is a loader for the JavaScript instead of a holder for the HTML resulting from the modules.

When an item needs to be represented in a view for which the layout is not yet known at the client-side, the application will detect if it misses the object description and will do a request at the server to get this. This technique is called lazy-loading and has been made possible with the work of again one of the community members (Doug Hendricks) and a small change to the core of the Ext JS library.

For some reason Ext JS does not offer a method to test if a certain object has been defined already. A method that would do this was easy to create. However the requested attribute that contains this information was made private, which made it impossible to access with an extension. A change to the core of Ext JS was therefore required to add this functionality. This was the only piece of functionality that could not be implemented with an extension to the Ext JS library.

8. Implementation

After the sfExtjsThemePlugin was getting more mature, the development of the intended application became relatively easy. The design, as shown in the sketches from section 4.6, could be implemented. The layout was defined with the use of an Ext JS viewport object, which fills the entire browser window. This viewport was initially subdivided in panels as designed in the sketches, but during the development process some changes were made. In this section the implementation of the application together with some of the differences with the initial design will be highlighted.

8.1 Navigation

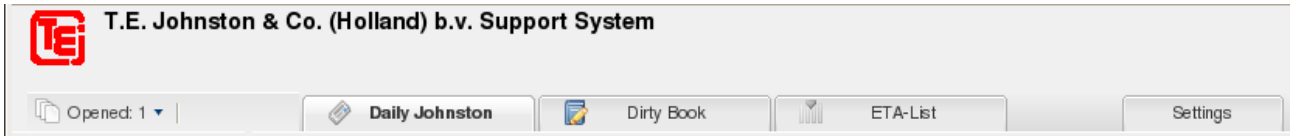


Figure 8.1: An overview on the complete top navigation

As designed, the navigation at the top (see figure 8.1) is divided in three parts. The first part contains a button showing the number of opened assignments and is connected to a drop-down menu showing an overview of these opened assignments. Next to it an optional close button can be seen that is visible when the content shows an assignment, which can be closed (see figure 8.2). This in contradiction to the overviews, that are not required to be closed.

Figure 8.2: The opened-assignments menu, together with the optional close button

The second part shows three tab-pages that provide access to the main overviews used at TEJ. These overview are: The Daily Johnston, the Dirty Book and the ETA-List. The first tab-page that is visible in the sketches from section 4.6 has been removed, since it was found to be confusing. However an overview at the left is replacing this, more about this below.

The third part of the menu contains only one tab-page that provides access to general overviews of the data in the system. In here all settings like properties of the company (like the name and address) and overviews from clients till cities can be found and modified. This however is not implemented at the moment.

As mentioned above, the general overview of all assignments, which could be accessed from the first tab-page in the menu, was replaced by a panel at the left-side of the application (see figure 8.3). This panel is always visible and shows an hierarchical overview of all voyages, with their assignments, with their parcels respectively. The active item in the content-panel (which can be a row in an overview or a detailed page) is automatically highlighted in the overview, to give the user an improved understanding on the data. The overview allows to browse and search through the data,

in order to find assignments and parcel. The detailed views on the parcels and assignments can also be opened from here and new assignments and parcels can be added, from the actions shown as buttons in the corresponding toolbars.

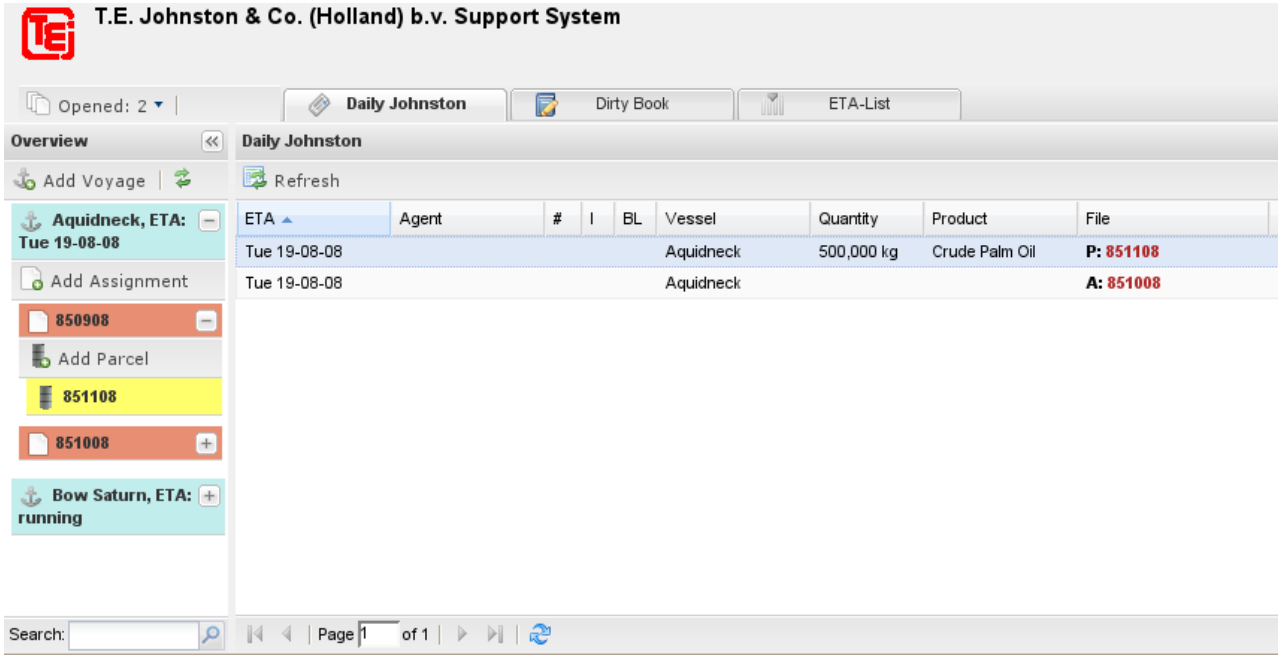


Figure 8.3: The general overview at the left that replaces the first tab-button in the top menu of the sketches, the menu-items are highlighted in correspondence with the content

8.2 Content

The content panel will contain the views that are generated by the administration generator. The initial view will be the overview of the Daily Johnston, represented by the first tab-page from the menu. The other menu entries from the top-menu will also appear in in the content-panel. The detailed views in their turn will show up after opening one of the items from the overviews. The fields in the detailed views sometimes refer to related items, as with cities that lay in a country. These relations are being used to support quick navigation from one detailed view to another as already shown in figure 6.11.

As can be seen in figure 8.3, the overview of the Daily Johnston does not contain any grouping of parcels per assignment like in the sketch at figure 4.2 any more. The grouping was implemented initially (a sample of this can be seen in figure 7.7), but the grouped headers merged several fields in one column, which made it impossible to sort the overview of any of these fields. Therefore it was chosen to implement the overview of the Daily Johnston exactly the way it was currently being used by TEJ in Excel. This makes much more sense and together with the permanent overview at the left of the screen all information about the hierarchy can still be seen.

8.3 Customising the Data Model

The overviews that are generated by the application show data combined from multiple entities. Not only the hierarchical data from voyages, that contain assignments that contain parcels are merged on one line, but also related values are combined in it, like client and vessel names. To generate these overviews efficiently the database should return this data with one query. This can be done by

writing custom peer methods in the model layer, in which the relationships can be defined. The current ORM is not smart enough to do this by itself and this is in fact the biggest development bottleneck when implementing the application. A big part of the hydration process needs to be written with custom code, which makes it again labour intensive and so worsens the maintainability. A solution to improve this is available in the form of Doctrine, the other ORM for Symfony.

8.4 Communication

The contacts overview that was designed at the bottom of the application has not been implemented. This is because all required functionality for this has not yet been implemented and besides, the old design might show the contacts a bit too prominent in screen. After playing around with the current application, a new idea is that it might be nicer to show a pop-up window that offers an overview of contacts, possibly assisted with an overview of the history of send messages.

9. User Feedback, Testing and Evaluation

Since this project is about developing usable business software, this project therefore relies heavily on interacting with the user and getting user feedback. Retrieving feedback is a must in order to prevent miscommunication and being able to deliver according to the expectations.

9.1 User Feedback

During this project the user was continuously involved in the development process. Initially by providing information about the company, later by providing feedback about the design, the mockup and finally the iteratively implemented application.

Multiple demonstrations have been given at the office, while the application was still in an unstable state, to show the working of the eventual application and to get feedback about alternative designs. Results from these demonstrations are for example the introduction of the new concept voyage, something that did not yet exist in the company's jargon (see section 4.1). Another result was the decision to not implement the designed grouped overviews of parcels under their assignment (as can be seen in figure 4.2 and as explained in section 8.2). Instead it was chosen to keep the overviews look exactly like the overviews that were currently being used at TEJ in order to have more control over the overviews.

Yet another important decision made after the demonstrations was not to keep track of the state of an assignment (see the note under “State aware edit-views” in section 4.5). It was found to be impossible to do reliably, since information about assignments could not be filled linearly. Besides, the functionality was found to be undesired.

During the implementation phase also a remarkable error was discovered. The invoices had been positioned under parcels (see section 4.6), while they were required under under assignments. This apparently had moved during the development process, to get noticed again after implementation had began. Because of the adaptability required to allow the iterative development style, the development environment fortunately had no problem of dealing with such a low-level design change.

Because of the open nature of the project, also feedback from other developers has been acquired. This has improved the tools as described in section 7.1.

9.2 User Testing

After good progress had been made with the implementation, several user test were performed with the real product. These tests should show how well the implemented application was capable of fulfilling the requirements of TEJ. These tests were performed as a field test [Schneiderman & Plaisant, 2005]: directly at the office, in the environment where the application should be running eventually. Since the system did not contain any data yet, it was not possible to look for information in the system when there was a phone call, but it would make the test more realistic. During the test an employee of TEJ was asked to perform several tasks, like adding a new assignment to the system, and find information about an assignment that had just been added to the system. To improve their understanding of the system, the employee was asked to look for specific information, while performing tasks to see the effects.

The actions of the user during the tests were recorded[10], to make it possible to review the handling of the user. This turned out to be useful, since actions that had been performed could be repeated in order to verify if identified issues were resolved. The recording of the third test unfortunately failed, due to technical reasons.

9.2.1 Test 1 - Monday August 11, 2008

The first test that was performed at TEJ was also the first time TEJ touched the developed application. It proved to be a valuable session where changes were proposed and some unresolved questions were answered, even after the extensive design phase. Also some small errors appeared to the surface.

A summary of the first test is given below:

Proposed changes

- Ex-tank should be an alternative for B/Ls, move its field to another tab-page
- Parcels are missing a field “load port”
- Contract depends on the product, it could be nice if this is automatically filled when a product is selected
- The text “description” for an analysis should be changed to “specification”
- You cannot add subjects of the B/Ls without uploading a file for the B/L
- Workflow support by tracking states of assignments and parcels not desired.

Errors

- Pop-up window for invoices does not fit in the screen
- Generating surveyor and laboratory assignments do not work properly for newly added parcels

Questions answered

- Assignments and parcels move from the Daily Johnston to the Dirty Book as soon as there is an invoice added to the assignment
- Changes between shippers and receivers require extra text for the report-templates
- Confirmed the Dirty Book contains the final weight, instead of the proposed weight as in the Daily Johnston
- The arrangement of fields in the edit-views was said to be logical

10 http://tejohnston.dynora.eu/tests/tests.html

Observations:

- During the test the combo-box was still in a somewhat "unstable" state. At the moment it tries to the user to be locked in the field until it contains a valid (or empty) value. This can introduce some problems when you want to change its value and will be fixed later on. During the test it required some knowledge about its behaviour, but did not introduce any problems.
- The button for adding new assignments (or actually first new voyages) was found immediately.

Figure 9.1: The "Add Voyage"-button was found immediately

- However, unfortunately the fields for entering the subjects of the assignments and parcels above the sub-tab-pages was not directly noticed. It might take getting used to, but another solution that can be tried is to move the general information to the first tab-page.

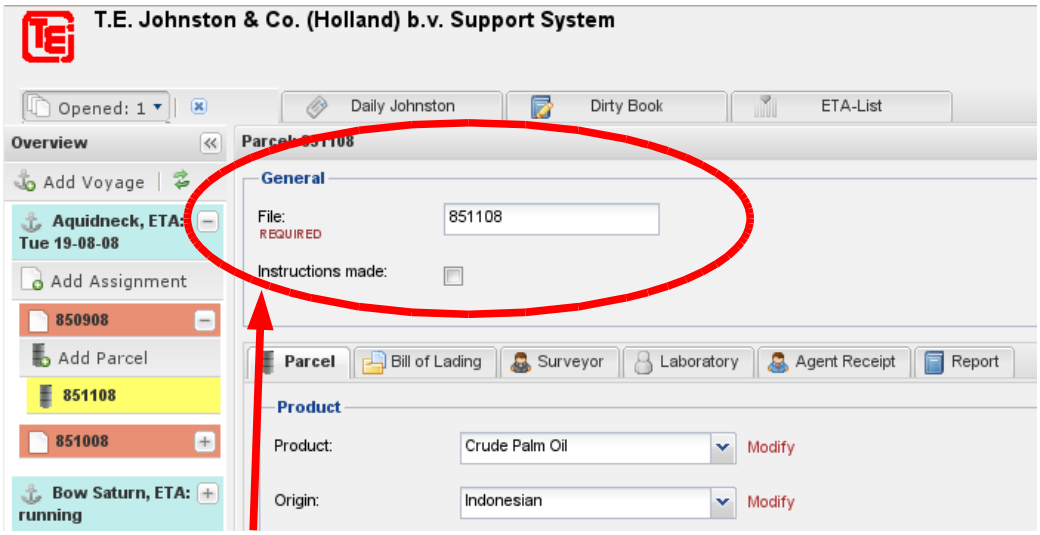


Figure 9.2: The "General"-area with the subject-field, above the tab-pages, was unnoticed

- Saving was done more than necessary. Although this does not cause any problems it could be improved. One solution can be to remove the save-button completely, saving changes automatically when a change has been made. This however introduces some other problems, for example when to save a new item that has required fields, which need to be filled in

before there can be saved. It would also be necessary to have a better and more obvious undo capability. Another problem is changing empty-fields, which are therefore not visible in the view-mode. A simpler solution to prevent the many saving will be to disable the save button, as long as there is nothing new to save.

- It was tried to open an assignment/parcel from the overviews by clicking on its corresponding line, instead of by clicking on the links in the subject columns.
- It would be nice to make it possible to narrow down the options in a combo-box when it contains values depended on another entity with an extra (chained) combo-box. For example narrow down cities by first selecting a country
- The origin of a product should be the adjective of a country, not the country name

9.2.2 Test 2 - Tuesday August 12, 2008

Only one day later the second test was already performed. Many improvements and fixes to solve issues of day one were already implemented. That was possible this fast because of the framework which allowed making changes easily.

Implemented improvements after test one

- The pop-up window that shows invoices now always fits on screen
- The generation of assignments for surveyors and the laboratory was improved
- The “ex-tank” field has been moved to the BL-tab-page
- A field has been added to define the “load port”
- Countries have a new property “adjective” to define the origin of a product
- The field “description“ has been renamed to “specification” for analysis
- Some fields have been rearranged
- New functionality have been added to allow the opening of voyages, assignments and parcels from the overview by double clicking on rows in an overview. Single clicking will be used for editing editable columns.

Questions answered

- It was asked if it would be nice to have columns that can be edited directly. It will not be an improvement to have all columns editable since this will increase the change of accidental changes. It turned out it would be nice to have the possibility to change the estimated quantity in the Daily Johnston and to be able to add new ETAs directly in the ETA-list, since this is done a lot and else you are required to open every voyage that has a new ETA, every Monday, Wednesday and Friday.
- More info about what feedback the surveyors and laboratory return will be provided

Observations

- The overview of voyages, containing assignments, containing parcels presented in the left panel is not used often. Maybe this is because all provided functionality is somewhat overwhelming, since the system allows to navigate in several ways. This overview however provides the only way for navigating to all the different elements of an assignment, where the other overviews only provide a link to the most detailed information (So to voyages, unless assignments have been defined, unless parcels have been defined). A solution can be to combine the fields of the three elements in the edit-views; providing voyages with tab-pages for every assignment, that in their turn contain tab-pages for every parcel. This will however also introduces some problems, like where to place the save-button(s). Optimally you would have only one save-button (or non at all as mentioned in section 9.2.1), but this would introduce technical difficulties: Saving will be done with overhead since all assignments and parcels will be transmitted every time (although this can be optimised as well), but more importantly all entities (voyage, assignments and parcels) should be separated and saved one after each other at the server side. Also handling of errors will become more difficult.
- Making changes to lists within edit-views for voyages, assignments and parcels will not require saving this is somewhat confusing. Also after the changes have been made the overviews like the Daily Johnston, the Dirty Book and the ETA-list are not automatically updated, which was even more confusing.

9.2.3 Test 3 - Friday August 15, 2008

During the third test the system ran stable; many errors had been fixed before this test and this showed. Only one technical problem showed up, and another problem was related to an unfinished template for generating reports. This was a good sign, since it was detected because the client could already find his way around this well around in the system, she was ready for the generation of reports.

The recording of this test unfortunately failed.

Implemented improvements after test two

- Ability to change estimated quantity directly in the Daily Johnston (adding new ETAs will require some more changes and will be done later)
- Making changes to lists in edit-views now automatically update the main overviews

Errors

- Generating the assignments for surveyors and parcels was broken in some situations. This turned out to be due to missing ETAs, which broke the entire generating-function, and has been fixed
- Survey-assignments are missing an overview of BLs

Questions answered

- The implementation of “Ex-tank” next to BL is approved

Observations

- BL should have a property weight, this was confirmed

Questions remaining

- What should be the default sorting in the assignment overview and the Dirty Book

9.3 User Evaluation

Even though not all requirements have been covered in the current implementation yet (with the biggest missing feature the ability to support communication), the tests could already show some very important issues regarding the use of the application.

Sometimes small changes were found to be required that could improve the usability of the application. For example the ability to open an assignment when a row in an overview is double clicked. This functionality had initially not been implemented, since there is a column containing a link that make this possible. However, since the employee was repeatedly trying to open assignments this way, it was clear this could improve the usability of the application.

Sometimes the tests also showed that there were bigger changes required to make the application meet the requirements. For example to fix the misplaced invoice field, that was wrongfully related to parcels, instead of assignments. This required changes from the data model all the way to the user interface, but because of the selected and extended development tools, changes like this could be made very fast and relatively easy.

At the moment the application can already handle most of the data that is being processed at TEJ, but before they can really start working with it, all of their requirements should be implemented. Currently not all reporting templates have been defined, processing the results of the surveyors and laboratory is not yet supported and as said communication is not supported either. After this has been implemented and TEJ has been working with the application for some while, a new test should be performed to find out if the currently noticed issues regarding the overview-panel at the left in the application and the overlooked properties at the top of a detailed parcel view have been solved or a alternative solution should be found. Even though the efficiency improvements that can be gained with the new support system already look promising, only at that time it is possible to perform a real user test to confirm this. For now TEJ is satisfied with current results and has enough confidence in the final result to continue the development after this graduation project.

10. Summary and Conclusion

Before coming to the conclusion of this project first a summary will be given.

10.1 Summary

The research question of this thesis is: *Is it possible to simplify the development of business software, without being restricted in possibilities, to be able to create custom business software that is usable and adaptable?*

To answer this question, information is gathered by trying to implement an application that is

- usable: the company should be able to efficiently do with its application what it requires
- adaptable: it should be simple to adapt the application to fulfil changed requirements

An iterative and user-centered development style was used, to create a support application for a company for which currently no support software was available.

To start this research project I therefore first had to find an appropriate company. This was found in T.E. Johnston & Co (Holland) b.v., a small survey company located in Barendrecht, that currently supported their activities with standard office applications. Chapter 2 describes the extensive study that has been performed on TEJ, required in order to design a usable application for them.

After the analysis had been performed, chapter 3 describes the three main requirements that have been identified, to which a support application for TEJ has to apply. These three requirements describe there should be one system that manages all data of TEJ. In order to get accepted and really support the company it should be usable; matching the work process at TEJ. Finally in order to make the application capable of becoming usable, an iterative development style has to be applied. This is required in order to handle miscommunication, as illustrated on the front page, and to be able to incorporate new requirements that will be identified during the implementation. Also since a company can change over time, it should be tried to make the application capable of dealing with this as well. That is why the latest requirement is to have the application adaptable.

In chapter 4 it is described how the analysis and requirements are combined to a design for their support application. The application should be capable of managing their assignments and support their work process. Sketches were made to illustrate how this would look like.

Chapter 5 describes the technical functionality that will be required in order to implement the design. However instead of directly implementing this functionality, tools and techniques are identified in chapter 6 that can be used to improve the maintainability when implementing this functionality. The tools that were searched should exploit the patterns that have been identified in the technical design to reduce the development time. The main techniques that make this possible are Object Relational Mapping and Administration Generation. These techniques were found combined in the Symfony Framework: A web server framework for PHP. Unfortunately the result was not interactive enough to meet the requirements. The client-side JavaScript framework was therefore added to the set of tools to solve this issue.

To combine the two frameworks and improve them somewhat more to meet all requirements they needed to be extended. This was an important task performed during this project, in order to make the final solution to an success. Chapter 7 describes what extensions were made to the identified

tools, in order to create a maintainable development environment that resulted in interactive applications.

The final application could now be created with the use of this extended tool. Since the tool was now capable of exploiting many of the repetitive patterns, seen during development of the application, implementation got relatively easy, as can be read in chapter 8. The resulting application should now be capable of managing assignments and showing overviews exactly as the employees were used to. Unfortunately support for communication and some other functionality has not been implemented yet, because of time constraints.

TEJ was actively involved during the entire development process, but to test if the resulting application was a usable solution for TEJ a number of user tests had to be performed. These are described in chapter 9. Even though the application was not completely finished yet, TEJ can already see how the application can reduce their workload and reduce the probability on making errors and is looking forward to start working with the final product.

10.2 Conclusion

Even though not all requirements of TEJ have been met in the implementation of their support system yet (the generator that has been written as a plugin to extend the available tools can "only" handle a big subset of the requirements at the moment), enough information has been gathered to answer the research question: *Is it possible to simplify the development of business software, without being restricted in possibilities, to be able to create custom business software that is usable and adaptable?*

To answer this question, it was mentioned the following sub-questions should be answered:

- Does the company approves the application? Since they are the one with the knowledge about the working of the company, they can tell if the application fulfils their requirements.

During the user tests it became clear the application still needs some work, but was already capable of requiring to enter data only once. TEJ could already work with a part of the application to understand its working. The implemented functionality made clear how the application can support their work process and was already found to be satisfying. They are therefore looking forward to start working with the final application, and in fact have already recommended it to several of their related parties.

- Is the implemented code really less complex?
  - is it faster to write
  - is it easy to write
  - is it easy to maintain

To test if the implemented code is less complex and easier to maintain a comparison has been made that can be found under "Improve Maintainability" in section 7.2.2. The exact speed improvement you gain when using administration generation when developing is hard to tell, but it can be clearly seen that much less work is required during implementation and the complexity has been reduced, allowing for an improved maintainability. This has already proven itself during implementation and the user tests, where new requirements could be fulfilled within a day.

It looks like the answer to the question: *Is it possible to simplify the development of business software, without being restricted in possibilities, to be able to create custom business software that*

is usable and adaptable? is therefore positive. To be able to give the final answer to this question, the developed application should get completely finished first. Then TEJ can work with it for some time, after which a new test should be performed to find out how well the application is supporting their work process and if it is really capable of improving their efficiency.

The reason that time constraints caused the application not to be finished, is mostly due to the lack of appropriate tools that were available. It seems that the current tools found were state of the art, since all frameworks that make use of it are currently only one or two years old. During this graduation project, a lot of time has been spend on extending the available tools, in order to be able to fulfil the requirements of TEJ. These extensions are combined in a plugin and have been made available as an open source solution. The result of this is that it is now being used in many different projects at different companies, in which it can be further extended to match different requirements.

The potential of the tools that have been found and extended look promising. The fact that the plugin is already being used at multiple companies to fulfil their specific requirements, while it was initially only designed to cover the requirements of TEJ, can only support this.

Before the application for TEJ can be finished, some remaining issues and missing functionality have to be solved. The most obvious requirement for which currently nothing has been implemented yet is the ability to support communication. Some other functionalities, like the ability to represent one-to-many relationships, is currently made possible using custom code. This should also be exploited by extending the generator even further. One last thing that would be wise to implement before the application for TEJ is going to be finished, is an improved solution for the model layer that currently still requires relatively many error-prone lines of custom code (see section 8.3).

10.3 Contributions

During this research project it became clear that companies have a desire for working with their own data, with their own workflow, to make it possible to improve their efficiency. Tools and techniques have been searched to make this possible. A technique called administration generation, that provides an abstraction layer for the development environment, makes this possible without reducing the potential functionality. An implementation of this technique was extended to make it more powerful (relations can more extensively be exploited and more option are provided to configure it) and the result more interactive (by replacing the output of static HTML by reusable JavaScript objects).

The current environment created supports agile development; capable of taking user feedback into account in combination with an iterative development style. The abstraction layer makes it possible to develop and maintain much faster and easier. This already showed during the tests, where a new version could be released within a day, while fulfilling changed requirements as deep as the model-layer.

Even though the extension has initially been developed to meet the requirements for TEJ it already showed that the solution is flexible enough to develop solution for other companies as well. At the moment several other companies make use of and improve the extension, among which the biggest is without a doubt the American Citi Bank.

10.4 Reflection

Software engineering is complex. Besides, the problem of developing a usable solution, another well known problem is the difficulty of estimating the amount of time that is required to implement functionality. When using a generator with powerful and functional templates, the amount of time to develop applications can be drastically reduced for repetitive tasks. This also makes it easier to estimate the amount of time required for implementation. However, once some required functionality cannot be fulfilled with the use of the generator, the development time to implement that functionality will also return to normal.

The iterative development and prototyping style applied can help in scheduling milestones and creating a shared understanding about the amount of effort that is required to create an application. As a downside the possibility of changing requirements increases, but this is for a big part under control of the user and is to serve the greater good: delivering usable business software.

Prototyping

To reduce the difference between the prototypes and the final application it would be ideal to make use of so called iterative prototypes, where the prototypes iteratively get transformed to the final product. This not only reduces the workload for the developers, since there is less overhead when developing the final product, but it also reduces the change on surprises for the users, since the final product will not differ much from the prototypes they have seen. This was not completely possible in this project since the tools for this were not available at the beginning of the project. With the developed plugin creating prototypes is much better possible since minimally configured modules already can give a good impression about the final result. Further refinements and the addition of custom code can transform the prototype in a final product.

If this was available at the beginning of this project, it would definitely reduce the difference between the prototypes and the final product. Not only in their visual resemblance, but also regarding the designed functionality. An example of this are the overviews that were designed in the prototype, where overviews were grouped on multiple fields. Since the prototype was not interactive at all it was not seen at that moment this would cause problems sorting and manipulating the data.

Generator functionality

At the moment there are still situations that cannot be automatically handled with the generator. These have currently been implemented with the use of custom code, since it was faster to write one exception with custom code then to extend the entire generator for it. However as soon as multiple similar situations showed up, it turned out it had been beneficial to transform the initial implementation to templates for the generator. Not only to reduce the development time for the reoccurring situations, but also to improve the maintainability and reduce the probability of errors. This again was some proof that the idea of administration generation can be of great help in reducing the complexity of implementation.

Community

Another nice thing that happened during this project, was me getting more an more involved with the Symfony framework. A result of this is that I am allowed to help with the improvement of the next administration generator within Symfony, which allows me to add my requirements directly to

the core and so get even more community feedback.

Also my work regarding the sfExtjsThemePlugin was mentioned and demonstrated during a Symfony conference by other people. This conference also contained presentation about other projects like the Doctrine project, which turned out to be stable at this moment and providing functionality that I was looking for to solve my last bottlenecks.

Future

Something completely different, but also very nice, was the announcement and release of browsers with new JavaScript engines. One of the problems with JavaScript is that it currently is being interpreted, which makes it relatively slow. These new browsers like Google Chrome and Mozilla Firefox 3.1 are compiling the JavaScript source, boosting performance by something like 40x the current speed, to make the application run just as fast as any other application developed in for example Java or C#. This will make the interface of the applications developed with the plugin described in this section even more responsive.

10.5 Future work

At the moment the application for TEJ already fulfils many requirements. However, there are still some points that need work. After the implementation of the application has been progressed far enough to make it able to fulfil the current requirements of TEJ, a new evaluation should be prepared to see how the application can improve the efficiency of the application.

Before the application for TEJ is going to be finished, it would be wise to first improve the administration generator with the latest knowledge that has been gathered to fix the current development bottlenecks that are known: the handling of one-to-many relationships and the amount of code and work that still is required in the model-layer (for handling joins and performing migrations).

After that the remaining requirements can be implemented, like the support for processing the results of the surveyors and the laboratory and the generation of all reports.

An detailed overview of the future work that has been planned can be found in appendix D.

11. Bibliography

Aiken, 2002: Peter Aiken, *ERP Considerations*, 2002

Bouwman, Hooff, Wijngaert & Dijk, 2005: Harry Bouwman, Bart vd Hooff, Lidwien vd Wijngaert and Jan A G M v Dijk, *Information & Communication Technology in Organizations*, 2005, ISBN: 1412900905

Davenport, 1998:Thomas H. Davenport, *Putting the Enterprise into the Enterprise System*, 1998

Ext JS Website: http://www.extjs.com/

Feldberg, 2007: Feldberg, 2007, *InterOrganizational Systems*, Dr Frans Feldberg

Hedman & Borell, 2003:Jonas Hedman and Andreas Borell, *ERP Systems Impact on Organizations*, 2003

Johannesson & Perjons, 2001:Paul Johannesson and Erik Perjons, *Design Principles for Process Modelling in Enterprise Application Integration*, 2001

JSON website: http://www.json.org/

Keuning, 2001: D. Keuning, *Bedrijfskunde*, 2001, ISBN: 9020731424

Laudon, 2002: Kenneth C. Laudon, *Management Information Systems - Managing the digital Firm*, 2002, ISBN: 0-13-061960-4

Lee, Siau & Hong, 2003:Jinyoul Lee, Keng Siau and Soongoo Hong, *Enterprise Integration with ERP and EAI*, 2003

Markus & Tanis, 2000:M. Lynne Markus and Cornelis Tanis, *The Enterprise System Experience - From Adoption to Success*, 2000

Markus, Petrie & Axline, 2001:M. Lynne Markus, David Petrie and Sheryl Axline, *Bucking the Trends: What the Future May Hold for ERP Packages*, 2001

Propel website: http://propel.phpdb.org/

Qooxdoo website: http://www.qooxdoo.org/

Ree, 2007: Leon van der Ree, *Enterprise Support Systems*, 2007

Reijswoud & Heuvel, 1997: Victor van Reijswoud and Willem-Jan van den Heuvel, *A Comparison between KISS and DEMO: Two novel approaches for information systems analysis in the Netherlands* , 1997,Proceedings of the 5th Annual Conference on Methodologies of the British Computer Society

Rouse, 1993:William B. Rouse, *Enterprise Support Systems*, 1993

Sakalos, 2008: Jozef Sakalos, *Writing a Big Application in Ext*, 2008, http://blog.extjs.eu/know-how/writing-a-big-application-in-ext/, last visit at 30-10-2008

Schneiderman & Plaisant, 2005: Ben Schneiderman and Catherine Plaisant, *Designing the User Interface*, 2005, ISBN: 0-321-26978-0

Soh, Kien & Tay-Yap, 2000:Christina Soh, Sia Siew Kien and Joanne Tay-Yap, *Cultural Fits and*

Misfits: Is ERP a Universal Solution?, 2000

Symfony website: http://www.symfony-project.org/

Themistocleous, Irani, O'Keefe and Paul, 2001: Marinos Themistocleous, Zahir Irani, Robert M. O'Keefe and Ray Paul, *ERP Problems and Application Integration Issues: An Empirical Survey*, 2001,

US Patent 7240070: Man Cheng, Kit - Levin, Issac Stephen - Ju, Wei-de, Dynamic generation of user interface components , 12/08/2005

Verville, 2003:Jacques Verville, *A Process Approach for Selecting ERP Software: The Case of Omega Airlines*, 2003

Wallace & Kremzar, 2001: Thomas F. Wallace and Michael H. Kremzar, *ERP: Making It Happen*, 2001, ISBN: 0-471-39201-4

Wikipedia - AJAX:http://en.wikipedia.org/wiki/AJAX,last visit at 30-10-2008

Wikipedia - ORM:http://en.wikipedia.org/wiki/Object-relational_mapping,last visit at 30-10-2008

Zaninotto, 2008: François Zaninotto, *RedoTheWeb - sfPropelFinder*, 2008, http://redotheweb.com/category/sfpropelfinder/, last visit at 30-10-2008

Zaniotto & Potencier, 2007: François Zaniotto and Fabien Potencier, *The Definitive Guide to Symfony*, 2007, ISBN: 9781590597866

Ziouvelou,Sideris & Nancy Pouloudi, 2004:Xenia Ziouvelou, Loannis Sideris and Nancy Pouloudi, *Network Management: Implementation Success Criteria*, 2004

12. Index of Figures

13. Glossary

Administration Generation A technique to automatically create user interface elements, based on the data model and a configuration file

Agile Agile software development makes it possible to do iterative development: continuously improving a product while requesting user feedback and taking this into account

AJAX *Asynchronous JavaScript And XML*: a combination of existing techniques which make it possible to create dynamic interactive web applications

Back-end At the back-end of an application the actions (from the front-end) and data are being processed

B/L *Bill of Lading*: A bond representing the value of a product

Controller An interface-layer that processes and responds to events between the view- and model-layer

COTS *Commercially, off-the-shelf*: 'standard' computer software for a specific business

CRUD *Create, Retrieve, Update and Delete*: the most common data actions performed on records in a database

DAC *Discretionary Access Control*: access control based on the owner state

Data Model A description of the data-entities and their relationships

DBMS Database Management System: a system that maintains a database

DRY *Don't Repeat Yourself*: a programming paradigm

ETA *Estimated Time of Arrival*

Ext JS A JavaScript Framework, containing many tools for creating webbased (office) applications

Fieldset A grouping of fields in a form

Front-end The front-end of an application contains the user interface, which allows interaction with the user and communicates with the back-end

Hydration The filling of an instance of a class with data

JavaScript A development language for web-browsers

JSON *JavaScript Object Notation*: a lightweight data-interchange format

KISS *Keep It Simple, Stupid*: a programming paradigm that states that unnecessary complexity should be avoided

LINQ *Language-Integrated Query*: a tool made by Microsoft to enhance its .Net environment, bridging the gap between datasources (like an RDBMS) and the development environment

LOC	*Lines Of Code*
Model	The layer in which the data structure is combined with domain logic
MVC	*Model-View-Controller*: a pattern to separate the logic from the user interface
MySQL	A database engine supporting foreign key constraints to automatically...
OO	*Object Oriented*: a development technique using objects to group basic variables and add behaviours to classes to make them describe and behave like real items
ORM	*Object Relational Mapping*: a mapping technique between a RDBMS and a development environment
Partial	a small template-file to write custom code in
PHP	*PHP Hypertext Preprocessor*: a development language supporting Object Oriented development, mostly used for web development
RAD	*Rapid Application Development*: a term that describes the speed up of development by using techniques, frameworks and iterative development.
RBAC	*Role Based Action Control*: access control based on privileges
RDBMS	*Relational Database Management System*: the today's standard for a DBMS
RoR	*Ruby-on-Rails*: a framework for Ruby
Ruby	A relatively new development language, intended to be simple and productive. In Ruby everything is an object, even primitives
Propel	An implementation of an ORM for PHP5
Scaffolding	The generation of an interface to support CRUD per item of the data model
SG	*Specific Gravity*: the ratio of the density of a substance at a specific temperature and pressure
SQL	*Structured Query Language*: a language designed to access data in a (R)DBMS
Symfony	A PHP5 framework supporting RAD
View	A layer to represent data, also allowing for ways to interact with it
WC	*Word Count*
XML	*Extensible Markup Language*: a generic purpose language that allows to represent structured data by surrounding items with tags and brackets
YAML	*YAML Ain't a Markup Language*: YAML is a syntax to represent data with, designed to be useful for human writing and reading. Unlike XML it does not use any tags and brackets, but simply aligning with (white)space to represent data structures
YML	See YAML

Appendix A. Inventory

- 1 Server
  - Disk size: 400GB
  - OS: Ubuntu Linux
  - Mail-server: Postfix + Dovecot
  - Webserver: Apache + PHP5
  - WebMail: Squirrelmail
  - FileSharing: Samba (Windows shares) + wu-ftpd
  - Backup: Online (Unison sync to two outside servers)
- 3 workstations + 2 laptops
  - OS: Windows XP
  - Webbrowser: Firefox (3.0)
    - Online banking
    - Used for currency conversion: Google
    - Used for translations: www.interglot.com
  - Mail: Thunderbird (iMap client)
  - Microsoft Office SB
    - Word
    - Excel
  - Unison on the laptops
    - to synchronise files with server for local copies.
- Internet
  - ADSL
    - provider: Xs4all
    - address: tejohnstonnl.demon.nl
    - services: ssh, ftp, http, imap, smtp
- Printer/Scanner/Fax
  - Canon Multifunctional black/white laser printer/fax/scanner
    - full duplex
    - document feeder
    - scans (double sided) to folder on server as pdf
  - Label/postage printer
  - HP Photo-printer
- Phones
  - 2 ISDN handheld
  - 2 VOIP handheld
  - 2 mobile phones

Appendix B. Data Model

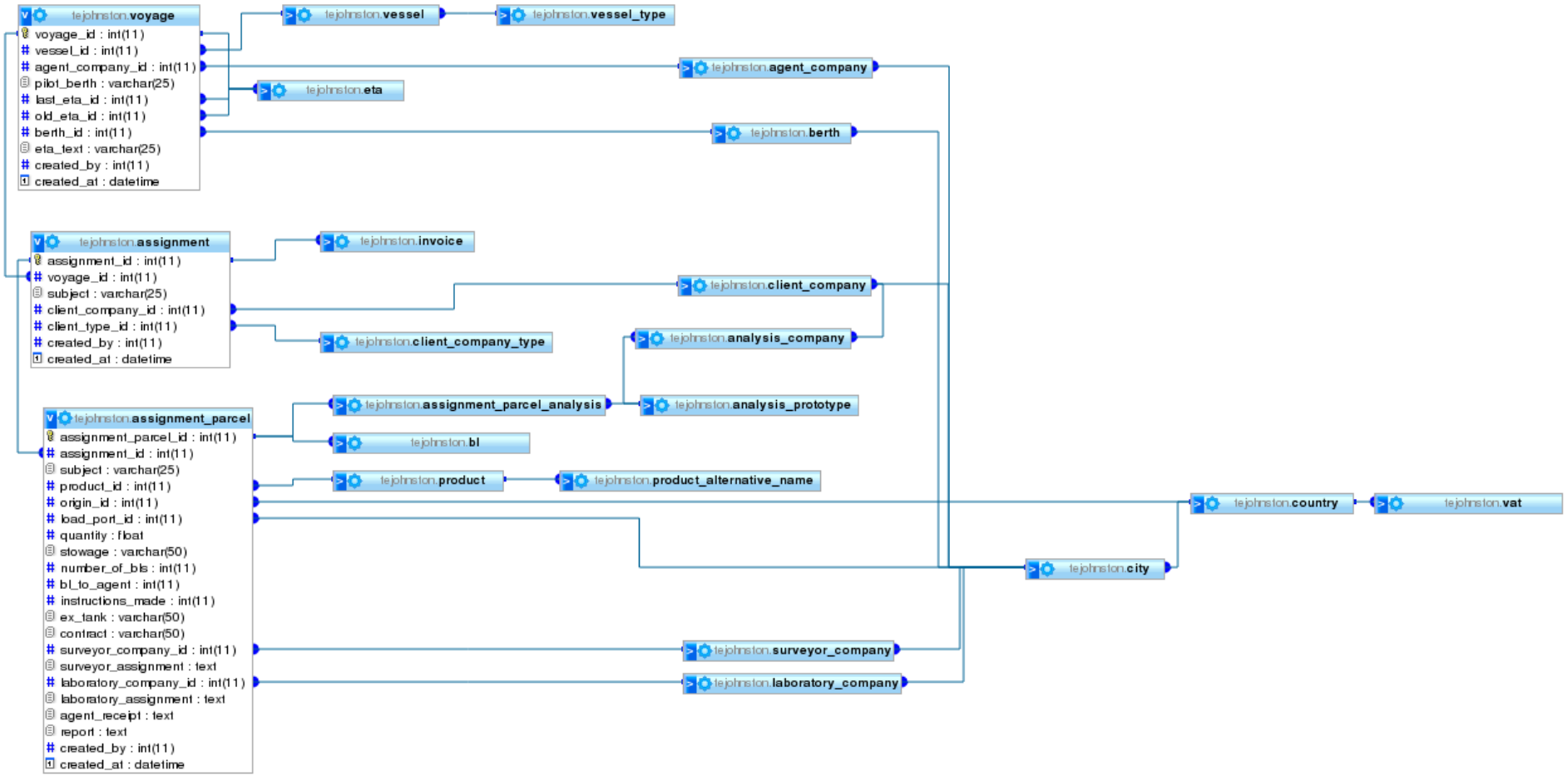


Figure B.1: A simplified version of the Database/Object relations. Some columns and tables are hidden for the sake of simplicity.

Appendix C. Tools

C.1 Webserver: Apache

Apache is an open source web server that runs on many platforms, like Windows, Linux and on the Mac OS. It is pluggable to allow you to set it up the way you require. In my case this meant I added the PHP-parser to it for my code and the Symfony-framework. Also a plugin to rewrite urls was enabled to allow nicer URLs.

C.2 IDE: Eclipse with PDT-tools

Eclipse is a multi-purpose Integrated Development Environment (IDE). Because it is pluggable it can be used to support multiple development languages, including PHP and JavaScript, and you are free to add extra tools to suite your needs, like for example a plugin to support SVN and even to execute some basic Symfony functions.

A couple of nice features of Eclipse with the PDT-tools plugin (to support PHP web-development) is that it support syntax checking, code completion and code-debugging. Another plugin was used to do the same thing for the JavaScript I wrote.

C.3 PHP Debugger: Xdebug

Xdebug is a powerful PHP-extension that allows for server-side debugging. Together with the PDT-tools for Eclipse this makes it possible to execute PHP-code step-by-step and watch variables being manipulated.

C.4 JavaScript Debugger: Firebug

Firebug is a plugin for Firefox (a webbrowser which runs on almost any platform and tries to follow web-standards) that allows for debugging of JavaScript at the client-side.

C.5 Database Management: PhpMyAdmin

PhpMyAdmin is a widely used webbased database administration tool for MySQL. I used it to test queries and to check the results of ORM-actions.

Appendix D. Detailed overview on future work

Developer Framework:

- Add an module manager allowing better interaction between different modules *(This will be a feature in the new administration generator of Symfony)*
- Exploit more patterns with the generator, like one-to-many relationships (at the moment this is done with additional custom code) *(This will also be a feature in the new administration generator of Symfony)*
- Credential checking and data validation at model level, this is currently done at module level *(more people in the community have this same desire and this will probably be made possible together with them)*
- Support migrations on the data *(This apparently is already supported by the ORM Doctrine)*
- Add a template editor for the reports (if possible usable directly for the users)
- Use another ORM-library that is more powerful and easier to use, making it easier to define complex queries, reducing development time and further improving maintainability *(possibly the dbFinder plugin if providing enough functionality, currently under active development [Zaninotto, 2008])*
- Reduce the learning curve for new developers, simplify the start-up and add a GUI-editor to configure the module-configuration and data model. This will provide additional testing and feedback from the community and improve development speed

User Usability:

- Improve the combo-box implementation to improve usability
  - Narrow down options when there are dependencies on other entities *(chaining, implemented for a big part already by Benjamin Runnels)*
  - Rethink about forcing to stay in the field when it contains illegal values, this can probably be improved by using visual feedback instead of the lock-in
- Show difference between editable and non-editable columns in overviews
- Implement communication
  - Allow direct mailing of information from the application
  - Show easy contact information
- Add view-screen next to edit-screen for details of entities
- Fix print layout *(plugin for Ext JS available)*
- Add non JavaScript theme (make it work at more computers) *(becomes easier with the new administration generator of Symfony)*
- Implement data pushing from server to keep user updated with latest data

- Improve usability in browser, for example by making use of the browsers back/forward button, and implement listeners to key-combinations *(natively available in latest Ext JS release)*
- Implement undo-capabilities from the server (time-machine), with (propel-)behaviours and a user interface to support it
- Implement support to allow eliminating of duplicates
- Allow to filter the overviews *(filters already available from the generator thanks to Benjamin Runnels again, only needed to be configured and added to the user interface)*.