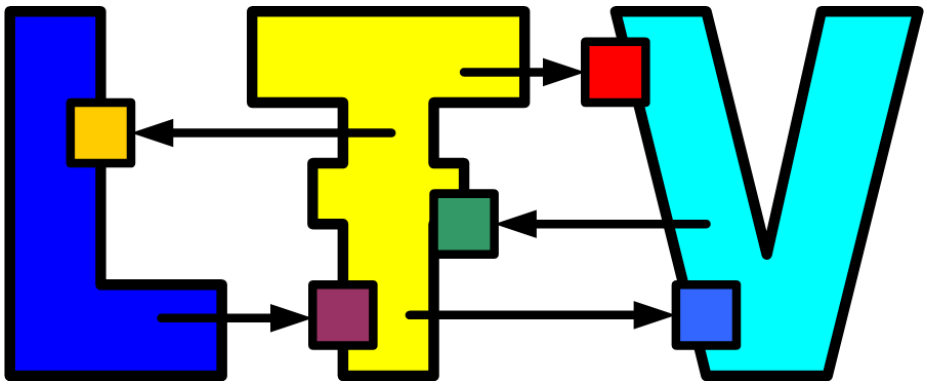
Diplomarbeit

REKONSTRUKTION UND ANIMATION AUS LOGFILES

LogFileVisualizer



Roland Fehlmann, Rico Steffen

Betreut durch Herr D. Keller

Geschrieben an der
HSR Hochschule für Technik Rapperswil
Oberseestrasse 10
8640 Rapperwil

Industriepartner
Stöcklin Software AG
Neuhofstrasse 5
8645 Jona

Autoren
Roland Fehlmann
Rico Steffen

Betreuer
Daniel Keller

Experte
Rudolf Mattmann

Version 1.0
13. Dezember 2006

Aufgabenstellung

Die Stöcklin Logistik AG, ein international tätiges Unternehmen im Gebiet der Förder- und Lagertechnik, realisiert an spezifische Kundenwünsche angepasste Gesamtsysteme. Ein Unternehmen der Stöcklin-Gruppe ist die Stöcklin Software AG, welche Software für solche Systeme entwickelt. Im Logistikbereich wurde ein Standardsoftwarepaket, LAKOS als Lagerverwaltungs- und Kommissioniersystem entwickelt.

Im Rahmen der Diplomarbeit soll ein Programm realisiert werden, welches den Umgang mit Logfiles, welche von LAKOS generiert werden, erleichtern soll. Optimal wäre es, wenn, mithilfe der Logfiles und der erstellten Software, die im Lager abgelaufenen Prozesse nachvollzogen werden könnten. Das Programm soll anschliessend zur Analyse eingesetzt werden und dabei helfen, Fehlfunktionen des Lagers zu erkennen und zu lokalisieren.

Um die Aufgabe lösen zu können, muss zuerst ein Überblick über die Funktionsweise eines Lagers und seiner Software gewonnen werden. Insbesondere die Kommunikation zwischen den verschiedenen Systemen soll analysiert werden. Anschliessend soll die Software geplant und umgesetzt werden. Die genauen Anforderungen werden dabei zusammen mit Stöcklin Software erarbeitet.

Abstract

Das Ziel der Diplomarbeit war es, eine Applikation zu erstellen, welche Logfiles, die beim Betrieb von Lagertechnik-Anlagen der Firma Stöcklin anfallen, visualisiert. Die Applikation soll dabei die Möglichkeit bieten, im Lager stattgefundene Abläufe nachzuvollziehen. Damit kann sie die Entwickler von Stöcklin Software bei ihren Analysearbeiten unterstützen und eine Hilfe beim Dokumentieren der Anlagen sein.

Um die gestellte Aufgabe erledigen zu können, mussten wir uns zuerst mit den Anlagen von Stöcklin vertraut machen. Dazu erhielten wir ein Testsystem, auf welchem die Simulation eines Lagers lief sowie eine Einleitung und diverse Dokumentationen von Stöcklin Software. Nach der Einarbeitungszeit entschieden wir uns, ein Programm zu entwickeln, welches die Telegramme, die zwischen zwei verschiedenen Schnittstellen ausgetauscht werden, visualisiert und diverse Filtermöglichkeiten anbietet, um unterschiedliche Telegramme zu selektieren.

Danach ging es an die Realisierung. Wir erstellten einen Parser, welcher die Logfiles einliest und in eine Datenbank schreibt. Über eine grafische Benutzeroberfläche erhält der Anwender die Möglichkeit, Filter auszuwählen und zu konfigurieren. Knackpunkte bei der Entwicklung waren die schlecht strukturierten Logfiles, welche das Einlesen erschwerten, die Filter, welche zum Teil ein gutes Verständnis der zugrundeliegenden Abläufe erforderten sowie das Finden einer geeigneten Darstellung, welche die Telegramme in einer übersichtlichen Art präsentiert.

Entstanden ist eine hilfreiche Applikation, welche die Erwartungen der Stöcklin-Entwickler erfüllt. Natürlich besteht noch Entwicklungspotenzial, so könnte eine grössere Konfigurierbarkeit an neue Anlagen verhindern, dass der Sourcecode der Anwendung angepasst werden müsste. Da die Applikation jedoch hauptsächlich von den Entwicklern verwendet werden wird, dürften solche Anpassungen kein Problem darstellen.

Management Summary

Ausgangslage

Die Firma Stöcklin Software hatte den Wunsch, eine Applikation zu bekommen, welche Logfiles visualisieren kann. Diese Software soll die Logfiles von zwei Schnittstellen so darstellen, dass die Entwickler von Stöcklin Software die im Lager stattgefundenen Abläufe nachvollziehen kann. Die Kommunikation zwischen den Systemen findet über Telegramme statt, diese Telegramme mussten wir sichtbar machen.

Zur Zeit existiert ein Tool, welches zwar die Logfiles nach bestimmten Inhalten filtern kann, aber keine Zusammenhänge darstellt. Die Logfiles werden auch nur im Textformat ausgegeben. Unser Ziel war es, eine Anwendung mit grafischer Benutzeroberfläche zu erstellen, welche in der Analyse und zu Dokumentationszwecken zum Einsatz kommen kann. Sie sollte möglichst einfach an neue Lagertechnik-Anlagen anzupassen sein.

Vorgehen

Der erste Schritt unserer Diplomarbeit war die Einarbeitung in die Welt der Lager und Paletten. Dokumentationen mussten studiert, Abläufe analysiert und Schnittstellen verstanden werden. Abbildung 0.1 zeigt als Beispielablauf den Start eines Verschiebewagens, welcher ein Teil eines Lagersystems ist. Dabei kommt vom Lagersystem der Befehl zum Kaltstart und der Materialflussrechener gibt die Befehle weiter an die SPS, welche die Hardware steuert. Die SPS ihrerseits liefert Statusberichte zurück.

Nach dem Einarbeiten ging es an den Entwurf der Software. Wir mussten ein User Interface entwerfen und die Architektur der Software festlegen. Dann definierten wir die Funktionalität, welche wir in drei Etappen implementierten. Am Ende jeder Phase wurde ein lauffähiger Prototyp erstellt. Die genauen Anforderungen an den zweiten Prototyp und das Schlussprogramm legten wir erst im Laufe der Arbeit fest – so konnten uns die zuständigen Personen von Stöcklin Software anhand der bereits vorhandenen Software mitteilen, welche zusätzliche Funktionen für sie wünschenswert waren.

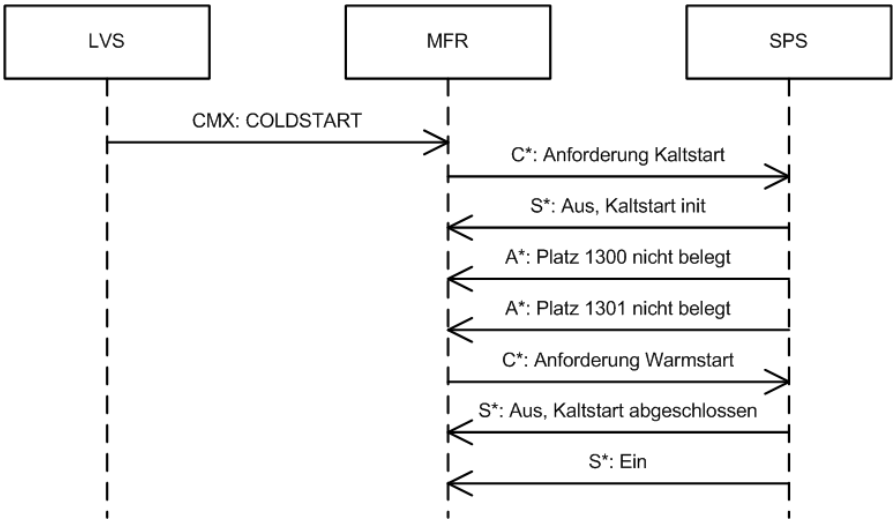


Abbildung 0.1: Ablauf eines Kaltstarts eines Verschiebewagens

Als der erste Prototyp fertig war, merkten wir, dass die von uns gewählte Darstellung der Telegramme nicht optimal war und wir passten sie für den zweiten Prototyp an. Auch die Art, wie wir die Logfiles eingelesen hatten, konnten wir verbessern: Wir entschieden uns, die Telegramme in einer Datenbank zu speichern. Auch neue Funktionalität kam in dieser Phase hinzu, wir implementierten eine Konfigurationsmöglichkeit per XML-Dateien und erste Filter.

In der letzten Implementationsphase implementierten wir alle zusätzlichen Filter und vervollständigten das User Interface. Zum Schluss wurde die Applikation getestet und letzte Anpassungen wurden vorgenommen.

Ergebnisse

Entstanden ist eine übersichtliche Applikation. Ein Screenshot ist in Abbildung 0.2 zu sehen. Es wird derselbe Ablauf gezeigt, wie er oben im Sequenzdiagramm dargestellt ist, diesmal aber aus dem Logfile ausgelesen und visualisiet. Die kleinen Unterschiede treten auf, weil die Telegramme asynchron verschickt werden und so die Reihenfolge nicht sichergestellt ist – aber auch nicht sein muss.

Die Applikation lässt sich über XML-Dateien konfigurieren. Die Konfigurationsmöglichkeiten sind jedoch nicht so flexibel, dass die angebotenen Möglichkeiten genügen würden, alle Spezialfälle neuer Lager-Anlagen abzudecken. Deshalb kann es sein, dass Anpassungen am Source-Code nötig sind. Der Programmaufbau in Kombination mit der Dokumentation sowie die Tatsache, dass die Applikation hauptsächlich von Software-Ingenieuren eingesetzt wird, sollten diese Aufgabe aber lösbar machen.

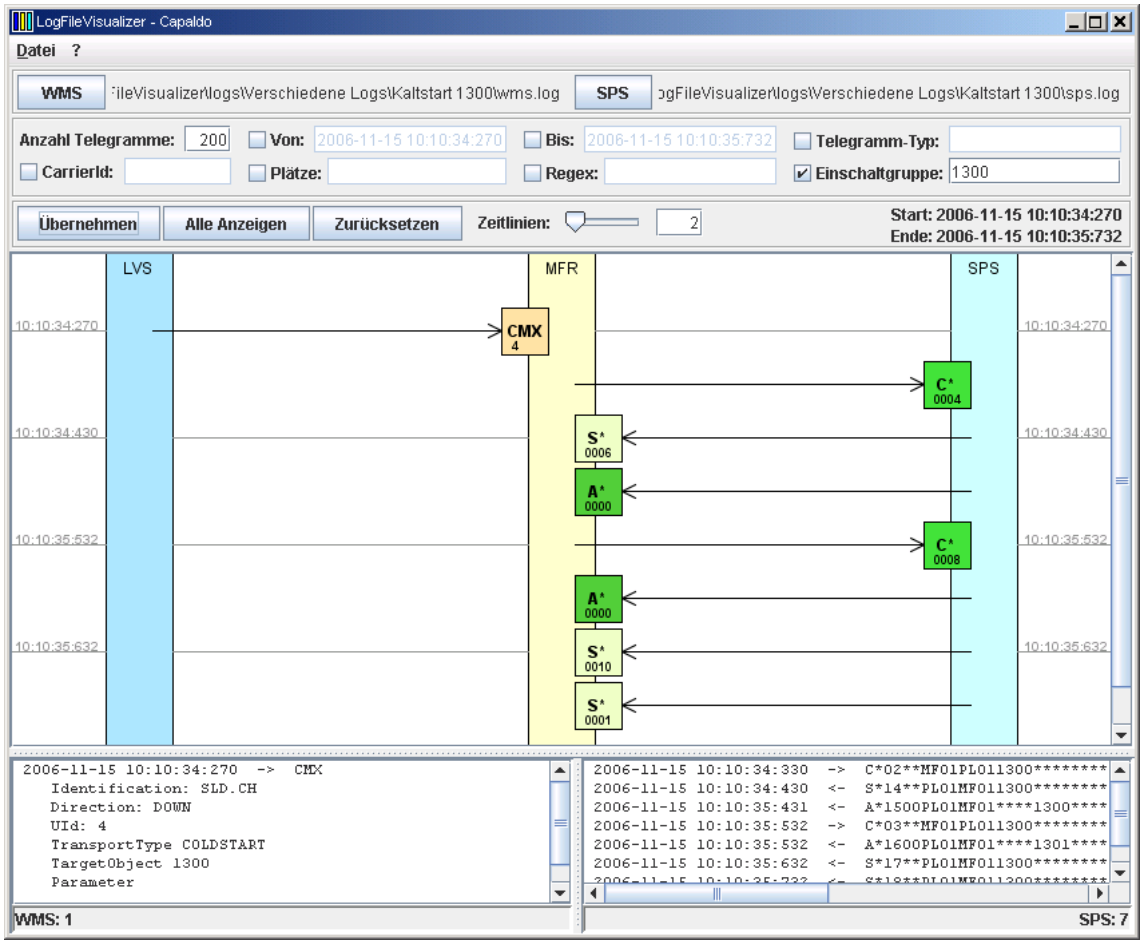


Abbildung 0.2: Screenshot unserer Applikation

Eine kurze Beschreibung der Applikation (von oben nach unten):

Menu Über das Menu können neue Logfiles und die Konfigurationsdatei eingelesen werden.

Logfile-Buttons Mit den Buttons können neue Logfiles geladen werden.

Filterbereich Oberhalb des Anzeigebereiches können verschiedene Filter ausgewählt und konfiguriert werden.

Telegrammanzeige Die eigentliche Visualisierung zeigt diejenigen Telegramme an, welche aus dem Logfile ausgelesen und von den Filtern selektiert wurden.

Logfileanzeige Zu unterst werden die gefundenen Telegramme im Logdateien-Format angezeigt, ähnlich wie sie im Logfile gespeichert sind.

Ausblick

Die entwickelte Software erfüllt die gestellten Anforderungen. Sie macht es möglich, Telegramme so anzuzeigen, dass man die Zusammenhänge erkennen und Abläufe nachvollziehen kann.

Erweiterungsmöglichkeiten gibt es vor allem im Bereich der Konfigurierbarkeit. Ein regelbasiertes System könnte es ermöglichen, alle notwendigen Anpassungen an eine andere Anlage per Konfigurationsdateien zu machen, ohne dass der Source-Code angepasst werden müsste.

Ein weiterer Punkt wäre die Integration des Tools in die bestehende Applikation zur Lagersteuerung von Stöcklin Software. Dank dem Einsatz der gleichen Technologien sollte dieser Schritt einfach zu realisieren sein. Eine Integration könnte es auch möglich machen, die Applikation mit dem bestehenden Visualisierungssystem zu verbinden, so dass man dann, anhand der Logfiles, die Kisten in der Lagersimulation “herumfahren” lassen könnte.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

I

Technischer Bericht

Verantwortlich: Roland Fehlmann

Datum	Version	Änderung
06. 12. 2006	0.1	Erstellt
08. 12. 2006	0.2	Einleitung
08. 12. 2006	0.3	Domainanalyse
08. 12. 2006	0.4	Ideen
10. 12. 2006	0.5	Resulate
11. 12. 2006	0.6	Umsetzung
11. 12. 2006	0.7	Fazit
13. 12. 2006	1.0	Final

1 Einleitung und Übersicht

1.1 Über dieses Dokument

Der Technische Bericht bietet eine Übersicht über die Diplomarbeit. Zu Beginn wird ein Überblick über die Aufgabe gegeben. Es folgt eine Einführung in das Gebiet der Lager und Paletten, insbesondere in die, für diese Arbeit zentrale, Steuerung der Lager und die dabei stattfindenden Abläufe. Anschliessend werden die erreichten Resultate präsentiert, bevor dann die Umsetzung erläutert und die interne Struktur der Software erklärt wird. Zuletzt folgt eine Auswertung: Projektverlauf, Entscheide, Statistiken werden dargeboten und ein Fazit wird gezogen. Auch ein Ausblick über mögliche Erweiterungen wird gegeben.

Die weiteren Dokumente der Gesamtdokumentation bieten einen vertieften Einblick und ermöglichen es, zu bestimmten Gebieten detailliertere Informationen zu erhalten. Die Dokumente im Einzelnen:

Projektplan	Informationen über das Projekt.
Anforderungs-spezifikation	Nichtfunktionale Anforderungen sowie Use Cases, umzusetzende Funktionen und eine Planung der zu erstellenden Prototypen.
Analyse	Die Systemarchitektur der Lagerverwaltungs-Software und Abläufe, welche im Lager stattfinden.
Architektur & Design	Der Aufbau der von uns erstellten Software sowie der Beschrieb von Design-Entscheidungen, welche im Verlauf unseres Projekts getroffen wurden.
Projekt-Monitoring	Analysen bezüglich dem Ablauf unseres Projektes: Statistiken über Zeitaufwand sowie erstellten Code und Klassen.
Test	Beschreibung unserer Testmethodik und von den durchgeführten Tests.
Persönliche Erfahrungen	Erfahrungsberichte, wie das Projekt, subjektiv gesehen, verlaufen ist.
Benutzerhand-buch	Anleitungen zur Installation, Bedienung und Erweiterung unseres Programms.
Anhang	Verwendete Software, Danksagungen, Glossar und Literaturverzeichnis.

1.2 Industriepartner

Abbildung 1.1: Förderanlagen

Unsere Diplomarbeit realisierten wir zusammen mit der Firma Stöcklin. Die Stöcklin Logistik AG[1] ist ein international tätiges Unternehmen im Gebiet der Förder- und Lagertechnik. Stöcklin realisiert Gesamtsysteme für Paletten- und Gebindestückgüter und orientiert sich dabei an den individuellen Kundenbedürfnissen. So entstehen innerbetriebliche Logistiklösungen in verschiedensten Branchenbereichen, welche nicht nur bezüglich Mechanik, Steuerung und Informatik reibungslos funktionieren, sondern auch besondere Produktgegebenheiten und betriebswirtschaftliche Ziele des Kunden berücksichtigen. Weitere Geschäftsbereiche neben der Förder- und Lagertechnik sind Container-Systeme und Flurfördermittel.

Die Stöcklin Software AG[2] ist ein Unternehmen der Stöcklin-Gruppe. Ihr hauptsächliches Tätigkeitsgebiet umfasst das Planen und Entwickeln von Software für Leitsysteme in der industriellen Automatisation sowie allgemeine technische Software als Anlagen- und Systembestandteil für den Wiederverkauf. Im Logistikbereich wurde ein Standardsoftwarepaket, LAKOS als Lagerverwaltungs- und Kommissioniersystem entwickelt und ist bei über 200 Endkunden erfolgreich im Einsatz.

[1] Stöcklin Logistik AG: www.sld.ch oder www.stoecklin.com
[2] Stöcklin Software AG: www.retis.ch

1.3 Projektidee

Im laufenden Betrieb einer Anlage werden u.a. sogenannte SPS-Telegramme zwischen dem Materialfluss-Teil des Lagerverwaltungssystems und den maschinennahen Steuerungen (SPS, i.d.R. Siemens S7) hin und her geschickt. Dieser Telegrammverkehr wird in einem (grossen) Logfile protokolliert.

Die Projektidee wäre eine Software, die diesen Telegrammverkehr analysiert und im optimalen Fall ein «Replay» eines bestimmten Zeitabschnittes animiert darstellen kann. Zur Fehlersuche wäre es extrem hilfreich, wenn ein Software-Entwickler quasi die «Video-Aufnahme» einer real abgelaufenen Szene von einer Anlage nochmals in Slow Motion anschauen könnte.

1.4 Aufgabe

Aufgabe war es, im Rahmen der Diplomarbeit eine Software zu entwickeln, welche die Logfiles einliest und visualisiert. Idealerweise so, dass, mithilfe des entwickelten Programms, die Aktionen, welche im Lager durchgeführt wurden, nachvollziehbar sind – damit das Programm bei Stöcklin Software in der Analyse eingesetzt werden kann.

Zuerst sollte ein Überblick gewonnen werden, wie die Steuerung der Lager funktioniert und welche Abläufe stattfinden. Bei der Erstellung des Programms musste beachtet werden, dass die Steuerung der Anlagen sehr spezifisch sein kann und je nach Anlage variiert. Ziel war es, eine einfach anpassbare und wo nötig konfigurierbare Software zu entwickeln.

2 Domainanalyse

2.1 LAKOS

In diesem Abschnitt wird ein kurzer Überblick über LAKOS gegeben. Eine detailliertere Beschreibung ist in Kapitel 16: *LAKOS Systemarchitektur* zu finden. Die Grundstruktur des Systems zeigt Abbildung 2.1.

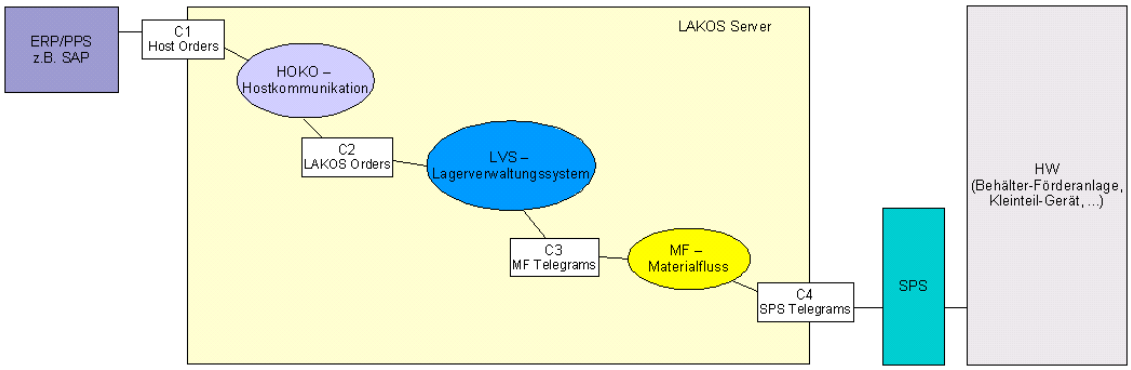


Abbildung 2.1: LAKOS-Struktur

Von einem ERP-System wie zum Beispiel SAP kommen Aufträge, welche vom HOKO in LAKOS-Orders umgewandelt werden. Das garantiert, dass LAKOS mit unterschiedlichen ERP-Systemen zusammenarbeiten kann. Das LVS wiederum koordiniert die Aufträge und gibt die entsprechenden Fahrbefehle an den Materialfluss weiter. Der Materialfluss seinerseits steuert SPSen an (Speicherprogrammierbare Steuerungen), welche die eigentliche Hardware, wie Förderanlagen oder Kleinteilgeräte, steuern.

2.2 Domainmodell

In Abbildung 2.2 sieht man eine Darstellung der Domänen-Objekte. Es wurde versucht, die logische Zusammengehörigkeit farblich hervorzuheben. Der blaue Teil umfasst die Logfiles und ihre Komponenten, das Lager und zugehörige Objekte sind rot eingefärbt und der Carrier ist grün. Nachfolgend werden die einzelnen Objekte kurz erläutert.

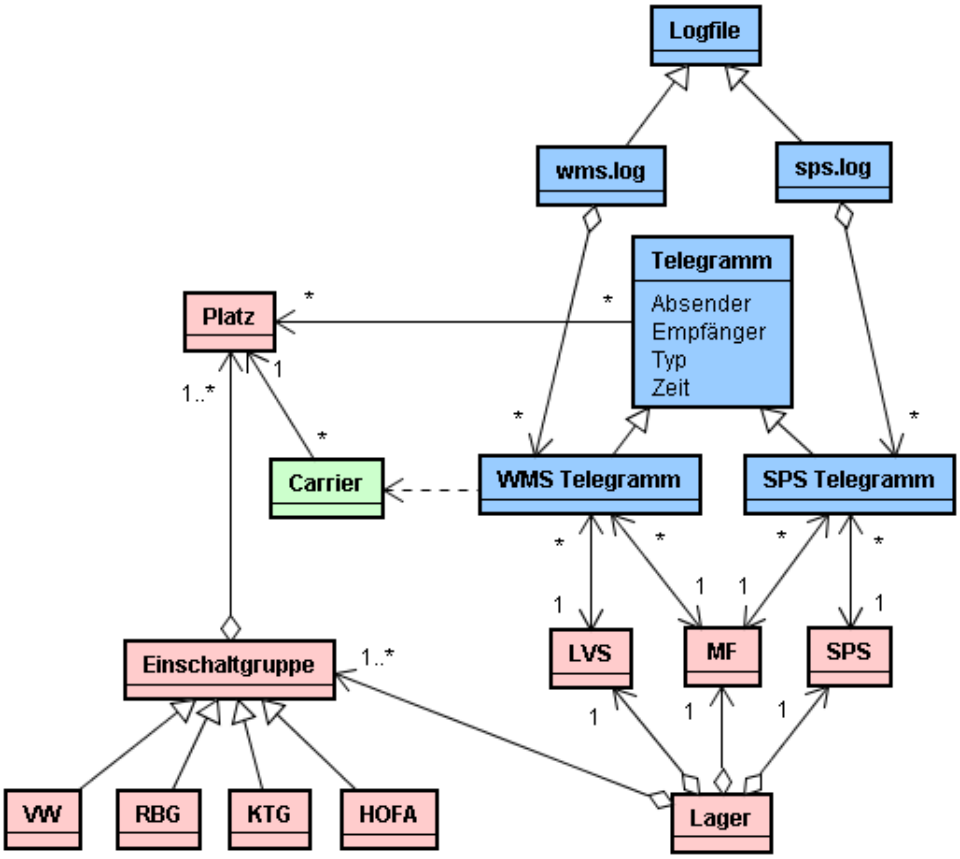


Abbildung 2.2: Domain Model

Logfile-Objekte

Logfile, wms.log, sps.log: Die eigentlichen Logfiles. `wms.log` stellt das Logfile der Schnittstelle zwischen LVS und Materialfluss dar, `sps.log` dasjenige der Schnittstelle zwischen Materialfluss und SPS. `Logfile` selbst repräsentiert das Konzept eines Logfiles, die beiden spezifischen Ausprägungen `wms.log` und `sps.log` werden davon abgeleitet.

Telegramm, WMS Telegramm, SPS Telegramm: Ein `Telegramm` stellt eine, im `Logfile` protokollierte, Meldung dar, welche zwischen zwei Systemen ausgetauscht wurde. Das `Logfile` setzt sich aus solchen protokollierten Meldungen zusammen. Ein `WMS Telegramm` ist eine Meldung, welche zwischen LVS und Materialfluss, ein `SPS Telegramm` eine, welche zwischen Materialfluss und SPS ausgetauscht wird. Beide Telegramm-Typen können bidirektional versendet werden.

Lager-Objekte

Lager, LVS, MF, SPS: `Lager` repräsentiert das komplette Lagersystem. Seine Steuersysteme, welche in unserer Arbeit von Bedeutung sind, sind das `LVS`, welches die eigentlichen Fahrbefehle absetzt, der `MF`, welcher die Bewegungen der Carrier koordiniert und die `SPS`, welche die Lagerhardware schliesslich steuert.

Einschaltgruppe, Platz, VW, RBG, KTG, HOFA: Eine `Einschaltgruppe` ist eine Gruppierung von mindestens einem `Platz` – auf einem Platz können ein oder mehrere Carrier stehen. Mögliche Einschaltgruppen können sein: Ein `VW`, Verschiebewagen, ein `RBG`, Regalbediengerät, ein `KTG`, Kleinteilgerät oder eine `HOFA`, Horizontalförderanlage.

Carrier-Objekt

Carrier: Ein `Carrier` ist ein, von der Anlage befördertes, Objekt. Zum Beispiel ein Palett, ein Karton oder ein Behälter.

2.3 Telegramme und Abläufe

Von zentraler Bedeutung in unserer Diplomarbeit sind die unterschiedlichen Telegramme, welche von den Systemen ausgetauscht werden sowie die Zusammenhänge, welche zwischen ihnen bestehen. Nachfolgend werden die wichtigsten Telegrammtypen beschrieben sowie ein Beispielablauf aufgeführt. Weitere Telegrammtypen, ihr genauer Aufbau sowie zusätzliche Abläufe sind in Kapitel 18: *Telegramme* beschrieben.

Wichtige Telegrammtypen

TRA-Telegramm Ein TRA-Telegramm beinhaltet einen Transportauftrag und wird vom LVS an den Materialfluss gesendet. Im Telegramm steht die Nummer des Carriers, der Startplatz (an welchem sich der Carrier typischerweise zu dem Zeitpunkt befindet, zu welchem das TRA-Telegramm geschickt wird) sowie der Zielplatz, an welchen der Carrier verschoben werden soll.

Z-Telegramm Z-Telegramme werden vom Materialfluss an die SPS geschickt. Sie initiieren eine Bewegung von einem Platz zum nächsten. Typischerweise enthalten Z-Zelegramme keine Informationen mehr

über den Carrier, welcher verschoben werden soll. Die Kontrolle, dass der richtige Carrier verschoben wird liegt beim Materialfluss. Es gibt unterschiedliche Arten von Z-Telegrammen (Z*, ZI, ZA), je nachdem was für eine Art von Hardware angesteuert werden soll.

M-Telegramm M-Telegramme sind Meldungen von der SPS an den Materialfluss, dass ein Carrier bewegt wurde. Durch sie erhält der Materialfluss das nötige Wissen, welcher Carrier sich wo befindet. Je nach Art der Hardware existieren unterschiedliche Arten von M-Telegrammen (M*, MI, MA).

RCX-Telegramm RCX-Telegramme werden vom Materialfluss an das LVS gesendet und können als Bestätigung angeschaut werden, dass ein Transportauftrag, welcher durch ein TRA-Telegramm aufgegeben wurde, durchgeführt wurde.

Error-Levels

Jedes Telegramm besitzt einen Error-Level, welcher anzeigt, in welchem Modus sich die Anlage befand, als das Telegramm gesendet wurde. So gibt es zum Beispiel `_INFO`-Telegramme, welche im Normalfall gesendet werden oder `TRACE`-Telegramme, welche zum Debuggen gesendet werden. Weitere Error-Levels sind `_WARN`, `ERROR` und `FATAL`.

Ein Beispielablauf

Diagramm 2.3 zeigt das Einlagern einer Kiste in der Anlage von Capaldo. Das Einlagern erfolgt nach folgendem Schema: Die Kiste wird aufgesetzt und vermessen. Die Destination erhält der Materialfluss vom LVS (durch ein TRA-Telegramm) und gibt anschliessend die entsprechenden Befehle an die SPS. Die SPS meldet jede Verschiebung mit einer Meldung (M-Telegramm) an den MFR. Die BR- und ED-Telegramme am Schluss des Vorgangs sind RBG-spezifisch und entsprechen in etwa Z- und M-Telegrammen.

Pärchenbildung

In einigen Anlagen kommt ein Mechanismus zum Einsatz, welcher sich Pärchenbildung nennt. Dabei fahren zwei oder mehr Carrier miteinander, stehen also immer gemeinsam auf dem gleichen Platz. Dadurch verfügt das Lager über eine höhere Kapazität. Auf die Steuerung wirkt sich die Pärchenbildung so aus,

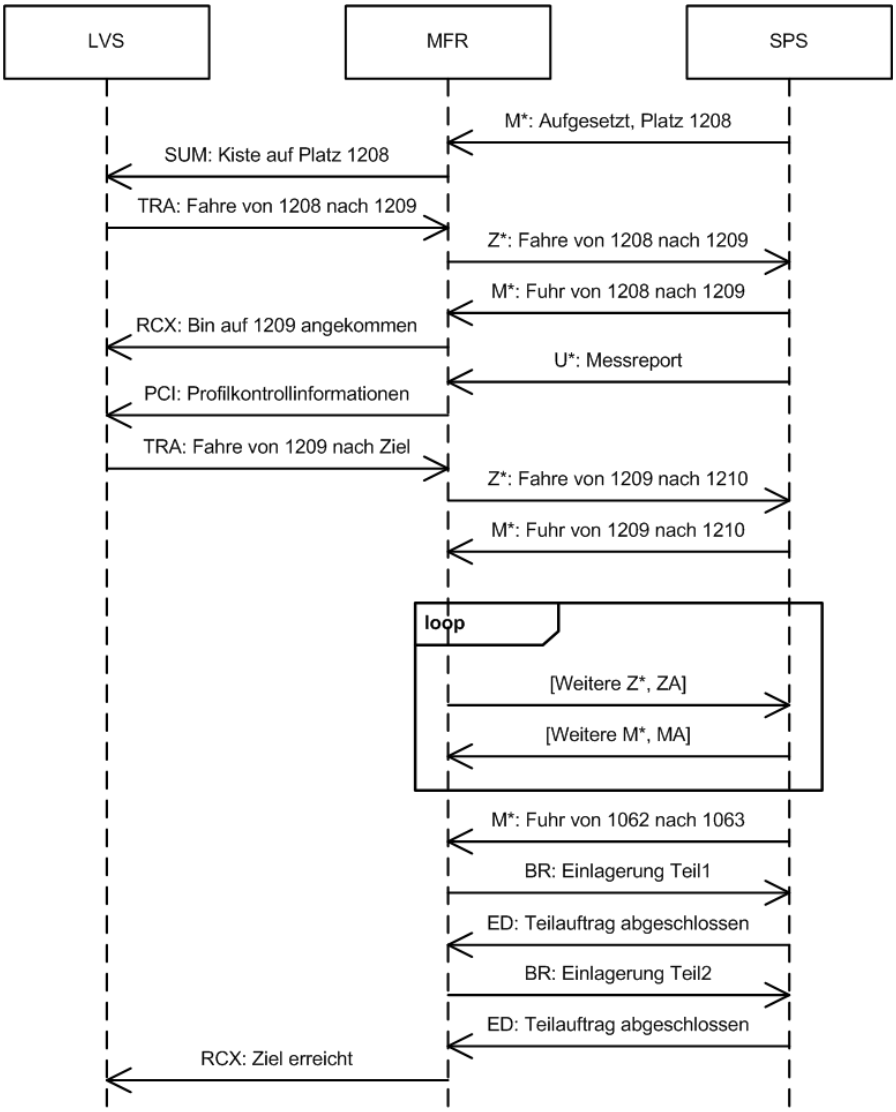


Abbildung 2.3: Kiste einlagern

dass, sobald ein Pärchen gebildet wurde, nur noch ein Z- und M-Telegramm geschickt wird, um die Carrier gemeinsam zu verschieben. Das erschwert die Zuordnung von einem Telegramm zu einem Carrier. Welche Arten von Pärchenbildung existieren und der detaillierte Ablauf sind in Abschnitt 18.3: *Abläufe* beschrieben.

3 Lösungsansätze und Ideen

3.1 Parsen und Filtern

Beim Parsen und Filtern hatten wir zwei Lösungsansätze. Der eine war das Vor- und Rückwärtsnavigieren im Logfile, so dass der Parser immer weiss, an welcher Stelle in der Datei er sich befindet und nach Bedarf weitere Telegramme in der gewünschten Richtung liefert. Die Filterfunktionen würden dann aus allen gelieferten Telegrammen die richtigen selektieren und zur Darstellung ausfiltern. Um eine angemessene Performance zu erreichen müsste wahrscheinlich ein Caching-Mechanismus implementiert werden, so dass immer genügend Objekte bereitstehen, welche dann dargestellt werden.

Der zweite Ansatz war das Speichern aller Telegramme in einer Datenbank. Das Selektieren der richtigen Telegramme könnte bei dieser Lösung über SQL erfolgen. Dies würde das Anwenden unterschiedlicher Filter relativ einfach machen. Nachteil bei dieser Lösung ist das vollständige Parsen und Speichern des Logfiles, was bei einem grossen Logfile ein bisschen Zeit beansprucht. Vorteile hat man hingegen durch den einfacheren Parsing-Vorgang, das Logfile muss nur einmal sequenziell gelesen werden und auch die Anwendung unterschiedlicher Filter dürfte sich einfacher gestalten, da man nur entsprechende SQL-Abfragen erstellen muss.

Entscheidung

Es wurde der zweite Ansatz implementiert. Im ersten Prototyp wurde noch der erste Ansatz favorisiert, danach aufgrund der Einfachheit jedoch gewechselt.

3.2 Ablauf

Auch beim Aufbau des Ablaufs vom Konfigurationsfile bis zur Darstellung hatten wir zwei Varianten. Dabei ging es darum, ob für SPS-Telegramme und WMS-Telegramme je ein separater Manager zum Einsatz kommen sollte oder ein Manager beide Typen handhabt. Es wurde nur ein Manager implementiert. Details zu den Designvarianten und dem Entscheid sind in Kapitel 20: *Design-Entscheide* beschrieben.

3.3 Darstellung

Die Abbildung 3.1 stellt eine erste Idee des Designs des User Interfaces dar. Diese Idee der Darstellung hatten wir bereits sehr früh und die Skizze konnte auch die bei Stöcklin Software zuständigen Personen überzeugen. Ein weiterer Gedanke, mit welchem wir uns befassten, war eine dreidimensionale Darstellung. Diese verwarfen wir aber wieder, da wir keinen gewinnbringenden Nutzen erkennen konnten und der Aufwand nicht zu rechtfertigen gewesen wäre.

Bei der gewählten Darstellungsart hatten wir im Sinn, vertikal eine lineare Zeitachse zu verwenden, die Abstände zwischen den Telegrammen also den Abständen der Sendezeiten der Telegramme anzupassen. Auch wollten wir die Telegramme so animieren, dass sie jeweils der Sendezeit entsprechend vom einen System zum anderen «fahren».

Entscheidung

Nachdem wir unsere Idee im ersten Prototypen umgesetzt hatten, merkten wir, dass die lineare Zeitachse nur viel Platz braucht, aber keinen Nutzen bringt. Auch die Animation behinderte die Arbeit mit der Software mehr, als dass sie etwas gebracht hätte. Deshalb implementierten wir eine simplere Anzeige, in welcher die Telegramme einfach chronologisch eingezeichnet werden.

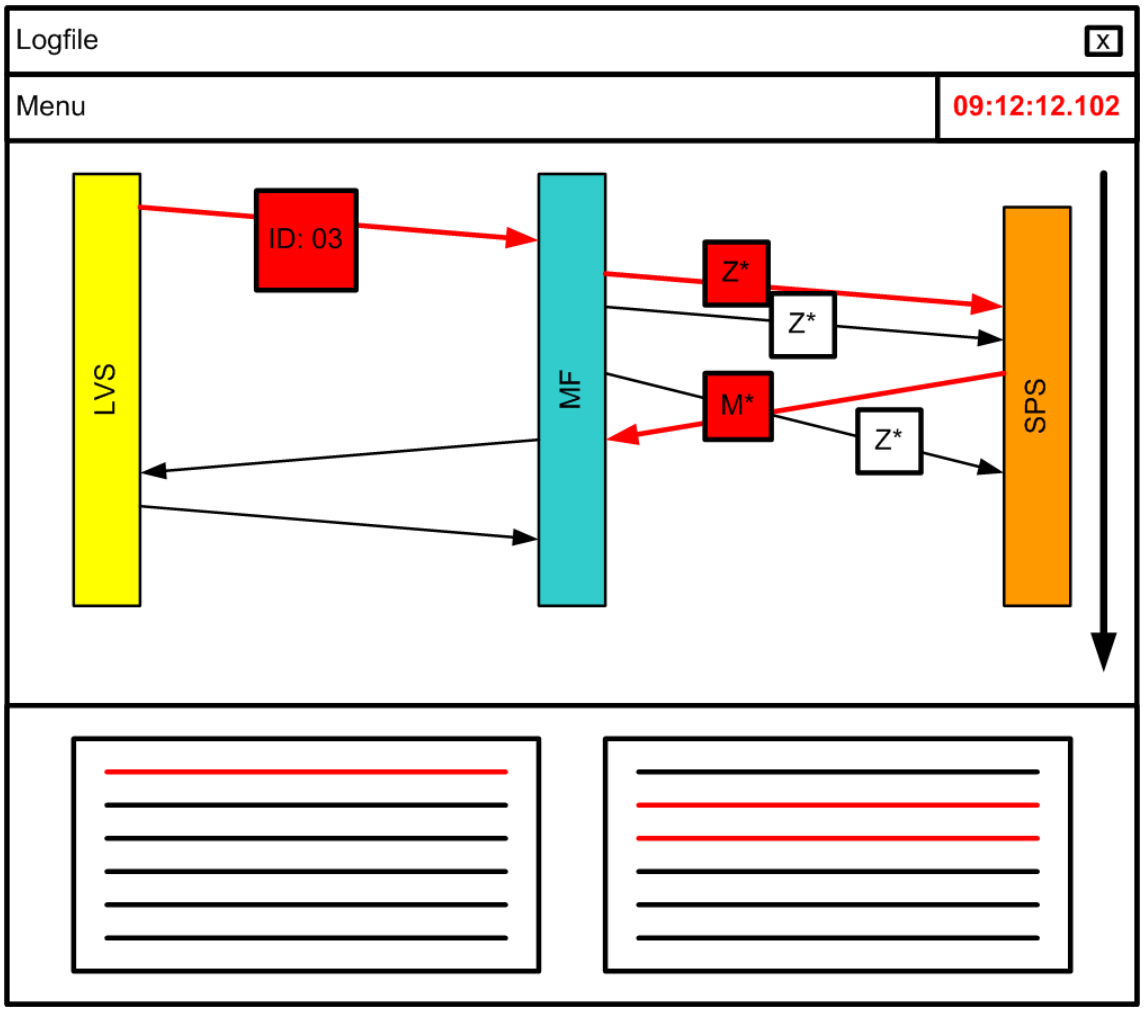


Abbildung 3.1: UI Skizze

4 Resultate

4.1 Screenshot

Abbildung 4.1 zeigt einen Screenshot des GUIs unserer Applikation.

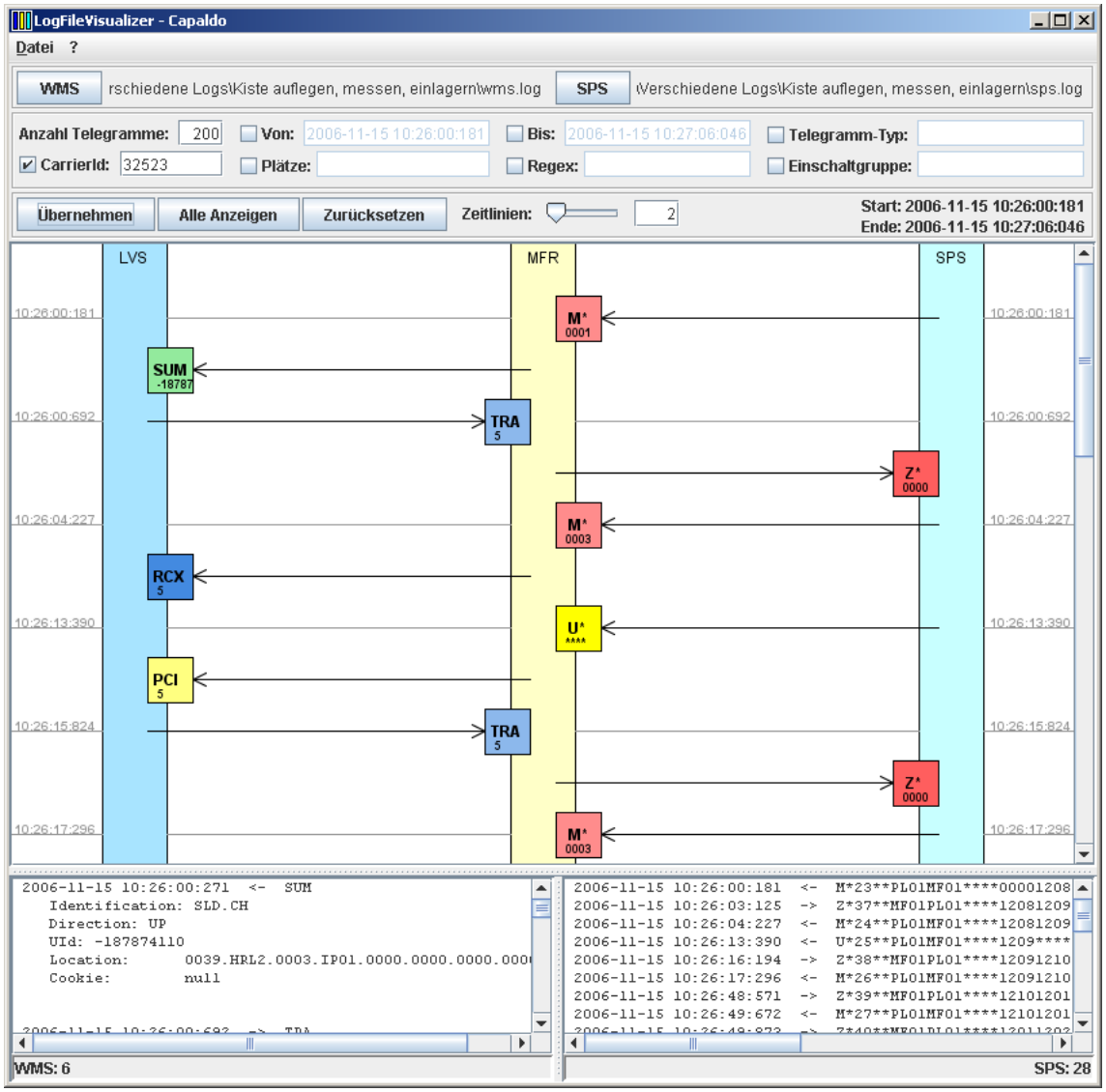


Abbildung 4.1: Screenshot unserer Applikation

4.2 GUI

Das GUI besteht grob gesehen aus vier Teilen: Dem Menu und den Buttons um die Logfiles zu laden; dem Filterbereich, in welchem die unterschiedlichen Filter selektiert und konfiguriert werden können; dem Anzeigebereich für Telegramme und der Darstellung der Telegramme in Textform.

Vom GUI aus können alle Funktionen der Applikation aufgerufen werden. Die Benutzung der Funktionen ist in Kapitel 27: *Bedienungsanleitung* beschrieben. Ein typischer Ablauf der Anwendung unserer Applikation sieht wie folgt aus:

1. Der Benutzer startet die Applikation und wählt ein zu analysierende WMS-Logfile sowie ein SPS-Logfile.
2. Der Benutzer wählt die gewünschten Filter, er möchte zum Beispiel die ersten 200 Telegramme ab einer bestimmten Zeit anzeigen. Durch «Übernehmen» startet der Benutzer die Anzeige der Telegramme.
3. Die Applikation zeigt die, den Filterkriterien entsprechenden, Telegramme im Anzeigebereich an. Zusätzlich werden die Telegramme in Textdarstellung ausgegeben.
4. Mit einem Klick auf ein angezeigtes Telegramm kann sich der Benutzer Detailinformationen über das Telegramm anzeigen lassen. Zusätzlich hat er die Möglichkeit, direkt aus dem Telegramm neue Filterkriterien zu extrahieren, so kann er zum Beispiel bei einem TRA-Telegramm die CarrierId auslesen und anschliessend nach ihr filtern.

4.3 Filter

Damit man sich nur bestimmte Telegramme anzeigen lassen kann, wurden diverse Filter umgesetzt:

Anzahl Telegramme	Ein einfacher Filter, welcher die Anzahl der angezeigten Telegramme auf die angegebene Anzahl beschränkt, indem er nur die ersten x Telegramme anzeigt.
Von, Bis	Erlauben es, nur Telegramme einer bestimmten Periode anzeigen zu lassen.
Telegramm-Typ	Filtert nach einem oder mehreren Telegramm-Typen, zum Beispiel TRA und/oder Z*.
CarrierId	Zeigt alle Telegramme an, welche sich auf einen bestimmten Carrier beziehen. So kann man sich zum Beispiel alle Telegramme anzeigen, welche gesendet wurden um eine Kiste einzulagern.

Plätze Filtert nach Telegrammen, welche sich auf einen bestimmten Platz oder mehrere bestimmte Plätze beziehen.

Regex Zeigt alle Telegramme an, welche den angegebenen regulären Ausdruck in ihrer textuellen Darstellung enthalten.

Einschaltgruppe Filtert nach Telegrammen, welche eine bestimmte Einschaltgruppe betreffen. Das ermöglicht es, sich die Kommunikation beim Einschalten bestimmter Lagerteile anzeigen zu lassen.

4.4 Konfiguration

Die Applikation bietet die Möglichkeit, sich über Konfigurationsdateien (Abbildung 4.2) an einzelne Anlagen anpassen zu lassen. Pro Anlage wird eine XML-Datei erstellt, in welcher die Definitionen der Plätze, an welchen Pärchen gebildet und wo sie aufgelöst werden, von welchen Plätzen aus ein- und ausgelagert wird sowie, welche alternativen Bezeichnungen für Plätze existieren.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="config.xsd">
<Facility>Capaldo</Facility>
    <PairBuildPlaces>
        <PullBuildPlace>
            <PlaceFront>1202</PlaceFront>
            <PlaceBack>1201</PlaceBack>
            <Code count="1">0001</Code>
        </PullBuildPlace>
        <PushBuildPlace>
            <Place>1205</Place>
        </PushBuildPlace>
    </PairBuildPlaces>
    <PairDismissPlaces>
        <AutomaticDismissPlace>
            <Place>1061</Place>
        </AutomaticDismissPlace>
        <ControlledDismissPlace>
            <Place>2090</Place>
            <Code count="1">0001</Code>
            <Code count="2">0002</Code>
        </ControlledDismissPlace>
    </PairDismissPlaces>
    <StorePlaces>
        <StorePlace nr="1063" zAxis="411" lane="006"/>
        <StorePlace nr="1062" zAxis="311" lane="006"/>
        <StorePlace nr="1073" zAxis="411" lane="007"/>
    </StorePlaces>
    <RolloutPlaces>
        <RolloutPlace nr="9060" zAxis="311" lane="006"/>
        <RolloutPlace nr="9061" zAxis="411" lane="006"/>
    </RolloutPlaces>
    <Aliases>
        <Alias name1="IP02" name2="1200"/>
        <Alias name1="PR02" name2="1202"/>
    </Aliases>
</Config>
```

Abbildung 4.2: Beispiel einer Konfigurationsdatei

5 Umsetzung

In diesem Kapitel wird dargestellt, wie das Programm umgesetzt wurde. Zuerst wird ein Überblick über die Architektur gegeben, anschliessend werden einzelne zentrale Teile genauer erläutert.

5.1 Architekturübersicht

Abbildung 5.1 zeigt eine Übersicht der Architektur. Danach werden die einzelnen Packages kurz beschrieben.

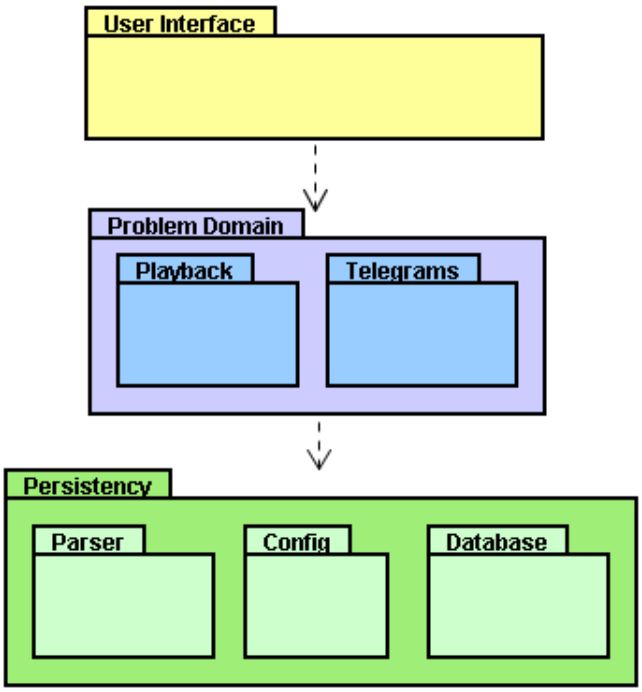


Abbildung 5.1: Architektur

User Interface

Das User Interface bildet die Schnittstelle zum Benutzer. Alle vom User durchführbaren Aktionen werden über das User Interface gestartet und zur Verarbeitung an die Problem Domain weitergegeben.

Problem Domain

Die Problem Domain beinaltet die eigentliche Programmfunktionalität.

Playback	Das Playback managt die Telegrammabfolge.
Telegrams	Enthält die Telegramm-Klassen.

Persistency

Die Logfiles werden durch die Persistency eingelesen und über eine Datenbank an die Problem Domain weitergegeben.

Parser	Der Parser analysiert die Logfiles und erstellt daraus die Datenbankeinträge.
Config	Liest die Konfiguration der Anlage ein.
Database	Erstellt die Datenbank und stellt die Abfrageschnittelle zur DB zur Verfügung.

Zusätzliche Packages

Im Diagramm nicht eingezeichnet sind die einzelnen Unterpackages sowie die Packages welche Hilfsfunktionen zur Verfügung stellen. Ebenfalls weggelassen wurden die Test-Packages.

5.2 Datenbank

Als Datenbanksystem kommt eine relationale Datenbank zum Einsatz. Die Datenbank besteht aus zwei Tabellen, welche in der Abbildung 5.2 aufgeführt sind. Die verschiedenen Telegramm-Typen werden in die entsprechende Tabelle geschrieben. Es besteht keine Beziehung zwischen den beiden Tabellen. Zur Performance-Verbesserung wird ein Index über die Spalte «time» geführt.

Bei den SPS-Feldern wurden teilweise deutsche Namen verwendet. Diese Bezeichnungen stammen aus der Dokumentation[1] von Stöcklin und wurden übernommen.

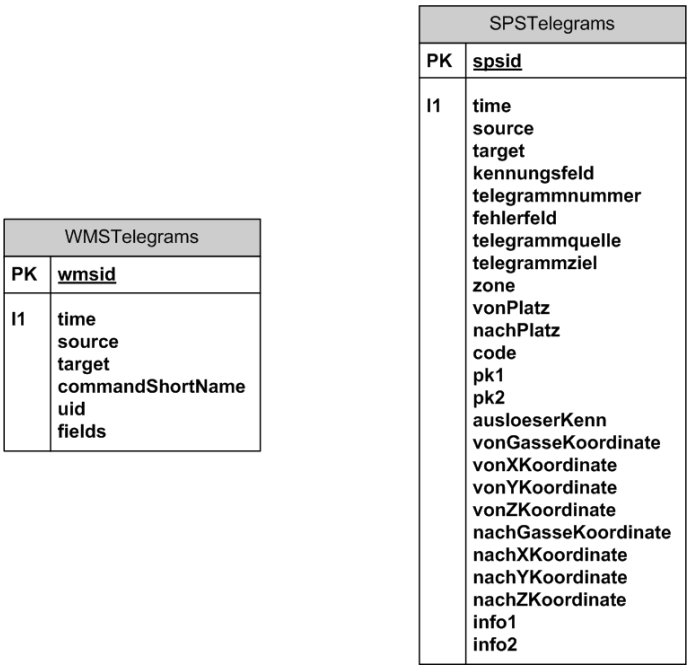


Abbildung 5.2: Datenbank

Der Zugriff auf die Datenbank aus der Applikation erfolgt per JDBC. Die Datenbank wird während des Programmstarts erstellt und danach beim Einlesen der Logfiles vom Parser gefüllt. Verwendet wird die Datenbank von den verschiedenen Filtern, welche jeweils die entsprechenden Telegramme abfragen. Bei Beendigung des Programms wird die Datenbank wieder gelöscht.

[1] Standard - Schnittstelle: Schnittstelle zu Subsystemen, Materialflussrechner / Lagersteuerungsrechner, Version 2.1

5.3 Parser

Der Parser hat die Aufgabe, den Inhalt der Logfiles in die Datenbank zu bringen. Für beide Logfile-Arten, das WMS- und das SPS-Log, existieren spezifische Parser, welche beide von der Basisklasse `Parser` erben und die jeweils unterschiedlichen Eigenschaften der Logfiles berücksichtigen. Beide Parser verfügen über einen `DataInserter`, welcher die geparsten Telegramm-Einträge übernimmt und in die jeweilige Datenbanktabelle schreibt.

Der `ParserThread` wurde gemacht, damit der Benutzer die Möglichkeit hat, den Parsing-Vorgang abzuberechen. Der Thread bekommt bei seiner Erstellung ein Objekt des Typs `ParsingCallback`, welches er über die Methode `finishedParsing()` benachrichtigt, wenn das Logfile fertig eingelesen wurde.

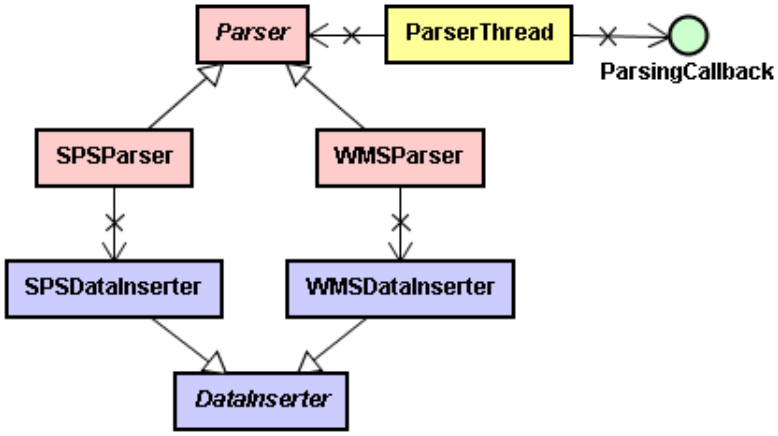


Abbildung 5.3: Parser

Die Parser wurden selbst implementiert, auf den Einsatz eines Parsergenerators wurde verzichtet. Dies, weil die WMS-Logfiles nicht wirklich formalisiert sind und weil es wahrscheinlich länger gedauert hätte, einen einsetzbaren Parsergenerator zu evaluieren, als den Parser selber zu schreiben. Die Abbildungen 5.4 und 5.5 zeigen ein paar Beispieleinträge aus den Logfiles. Es ist ersichtlich, dass das SPS-Logfile schön strukturiert und dementsprechend einfach zu parsen ist, das WMS-Logfile hingegen einen chaotischen Eindruck macht. Beim Parsen werden nur Telegramme des Error-Levels `_INFO` beachtet und Telegramme für die Visualisierung werden ignoriert.

```
2006-11-07 06:17:18:010 +000 INFO_            [SPSTargetSC08.SPSTarget.sendTopTelegramInOutputQueue:551]     ->
BR77**MF01SC08************ENDE**********0081500112010089999994111************************
2006-11-07 06:17:55:086 +000 INFO_            [SPSTargetSC08.SPSTarget.readDataAndPutIntoTheInputQueue:210]  <-
ED6400SC08MF01************0000********BR0081500112010089999994110001412111100899999999***
2006-11-07 06:17:55:305 +000 INFO_            [SPSTargetSC08.SPSTarget.sendTopTelegramInOutputQueue:551]     ->
BR78**MF01SC08************ENDE**********0089999994110089219212211************************
2006-11-07 06:18:32:491 +000 INFO_            [SPSTargetSC08.SPSTarget.readDataAndPutIntoTheInputQueue:210]  <-
ED6500SC08MF01************0000********BR0089999999411008921921221000174011110089999999999***
2006-11-07 06:18:32:506 +000 INFO_            [SPSTargetPL01.SPSTarget.readDataAndPutIntoTheInputQueue:210]  <-
M*46**PL01MF01****908020820003*************************************************************
2006-11-07 06:18:32:866 +000 INFO_            [SPSTargetPL01.SPSTarget.sendTopTelegramInOutputQueue:551]     ->
Z*67**MF01PL01****208220810000*************************************************************
2006-11-07 06:18:58:630 +000 INFO_            [SPSTargetPL01.SPSTarget.readDataAndPutIntoTheInputQueue:210]  <-
M*47**PL01MF01****208220810003*************************************************************
2006-11-07 06:18:58:849 +000 INFO_            [SPSTargetPL01.SPSTarget.sendTopTelegramInOutputQueue:551]     ->
```

Abbildung 5.4: Auszug aus einem SPS-Logfile

```
2006-11-07 06:26:48:391 +000 INFO_            [SPSLink.JMSLink.addMessageToQueue:164] adding Command to queue [MF_TO_WMS]:     DRM
    Identification: SLD.CH
    Direction: UP
    UId: 36632
    Location:      0039.HRL2.0003.C008.0000.0000.0000.0000

2006-11-07 06:26:48:547 +000 INVEN            [SPSLink.JMSLink.addMessageToQueue:168] adding Command to queue [MF_TO_VISU]:     VIM
    Identification: SLD.CH
    Direction: UP
    UId: 36632
  mforder carrierid 44715  mforder location  0000
2006-11-07 06:54:35:613 +000 INVEN            [ActiveMQ Session Task.JMSLink$1.onCommand:65] Received command from WMS:     CMX
    Identification: SLD.CH
    Direction: DOWN
    UId: 1369
    TransportType FULLPLANT
    TargetObject
    Parameter
```

Abbildung 5.5: Auszug aus einem WMS-Logfile

5.4 Filtern

Abbildung 5.6 zeigt die für das Filtern relevanten Klassen. Der `FilterHandler` ist zentral, da dieser die eigentlichen Filterfunktionen anbietet. Der `FilterHandler` wird vom `Manager` aus aufgerufen, welcher die Anfragen seinerseits vom User Interface erhält. Für die eingentlichen Telegrammabfragen steht dem `FilterHandler` die Klasse `DataSelector` zur Verfügung. Der `DataSelector` führt die SQL-Abfragen über JDBC auf der Datenbank aus. Spezifische Informationen über einzelne Plätze erhält der `DataSelector` aus der Konfiguration, über welche er via die Klasse `ConfigAccess` zugreift. Die gefundenen Telegramme werden in `TelegramQueues` zurückgeliefert.

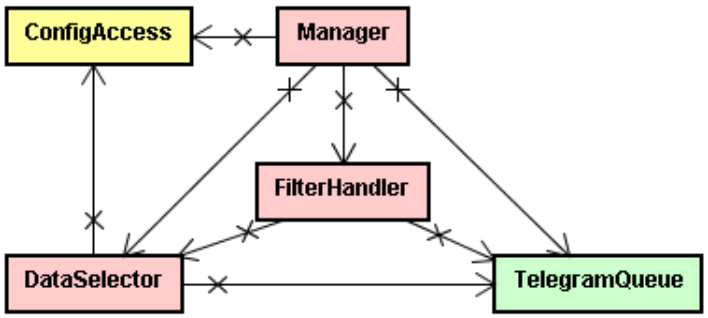


Abbildung 5.6: Filter

Der komplizierteste Filter ist sicherlich derjenige, welcher es erlaubt alle Telegramme eines Carriers anzuzeigen. Abbildung 5.7 zeigt ein Sequenzdiagramm, welches den Ablauf dieses Filters darstellt. Die `applyFilters`-Methode des FilterHandlers ruft eigentlich alle Filter auf, welche gesetzt sind, dargestellt ist aber nur der Ablauf für `trackCarrier()`. In dieser Methode wird zuerst nach einem beliebigen WMS-Telegramm gesucht, welches die angegebene CarrierId enthält. Von diesem wird die UId ausgelesen und es werden alle WMS-Telegramme selektiert, welche diese UId enthalten. Damit enthält man alle WMS-Telegramme, welche den Carrier betreffen. Danach geht es darum, einen Link zu der SPS-Seite herzustellen. Da in den SPS-Telegrammen meistens keine CarrierId gespeichert ist, wurde diese Verbindung über ein TRA-Telegramm realisiert. Das TRA-Telegramm bezieht sich immer auf einen Carrier an einem bestimmten Ort. Hat man ein TRA-Telegramm gefunden, kann man davon ausgehen, dass sich das nächste M-Telegramm, welches eine Verschiebung von diesem Platz weg indiziert, auf den gewünschten Carrier bezieht. Danach kann man den Carrier Platz für Platz verfolgen, da jede Bewegung durch ein M-Telegramm gemeldet wird.

Speziell sind noch die Übergänge von der HOFA zu einem RBG oder umgekehrt: Die BR- und ED-Telegramme vom RBG beziehen sich nicht auf einen speziellen Platz sondern müssen über die Gassen-Nummern und Z-Achsen-Koordinaten zugeordnet werden. Die entsprechenden Informationen bezieht der `DataSelector` via den `ConfigAccess` aus der Konfiguration der Anlage.

Alle `TelegramQueues`, welche der `DataSelector` dem `FilterHandler` zurückliefert, werden von diesem zu einer Queue zusammengeführt welche anschliessend dem `Manager` übergeben wird.

5.5 Darstellung

Das User Interfaces verwendet das Observer-Pattern. Es observiert den `Manager`, welcher jeweils Updates sendet, falls neue Telegramme anzuzeigen sind oder andere Dinge geändert haber, welche das User Interface darstellen muss. Das User Interface kommuniziert ausschliesslich mit dem `Manager`, dieser abstrahiert die Funktionalität der Applikation.

Als Technologie kommt Swing zum Einsatz, da auch das LAKOS damit erstellt wurde. Zur Anzeige der einzelnen Telegramme wurden Buttons eingesetzt, dies weil sie eine einfache Anbindung diverser Handler erlauben. So kann der Benutzer direkt auf die Telegramme klicken um eine Aktion auszulösen.

Werden neue Filter angewendet, zeichnet das User Interface die neuen Diagramme mit einer kleinen Animation; alle Telegramme fahren gleichzeitig von dem System, welches sie gesendet hat zum Zielsystem. Die Animation dient hauptsäch-

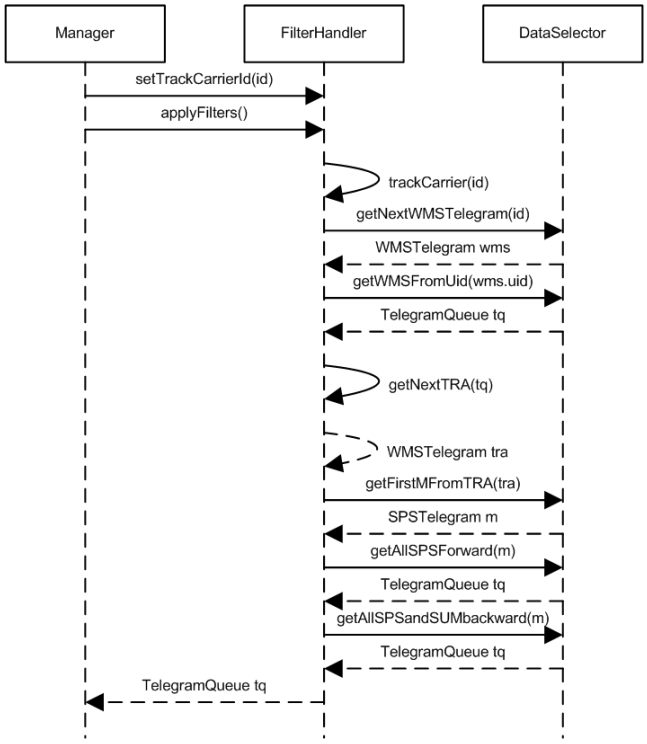


Abbildung 5.7: Ablauf des Filters zur Verfolgung eines Carriers

lich dazu, dass der Benutzer wahrnimmt, dass die neu selektierten Diagramme dargestellt werden und hat keine tiefere Bedeutung. Werden mehr als 200 neue Telegramme gezeichnet, wird aus Performance-Gründen auf eine Animation verzichtet.

Die maximale Anzahl der darstellbaren Diagramme wurde auf 2000 limitiert. Bei mehr Diagrammen treten Schwierigkeiten bezüglich des Speicherbedarfs auf, auch geht die Übersicht verloren. Nicht angezeigte Diagramme können aber einfach zur Anzeige gebracht werden, indem man einen Datumsfilter einsetzt.

5.6 Konfiguration

In der Konfigurationsdatei können anlagenabhängige Einstellungen gemacht werden. Das beinhaltet den Namen der Anlage, die Definition der Plätze an welchen Pärchen gebildet und wieder aufgelöst werden sowie die Ein- und Auslagerungsplätze der Anlage. Hinzu kommt die Definition von Aliasen für Plätze, welche im LVS eine andere Bezeichnung haben als im MFR. Als Format für die Konfiguration wurde XML gewählt, da die Dateien in dieser Sprache angenehm zu erstellen sind. Ein weiterer Vorteil ist, dass für Java diverse XML-Parser verfügbar sind,

welche das Einlesen auf einfache Art und Weise erlauben. Abbildung 5.8 zeigt das XML-Schema der Konfigurationsdatei.

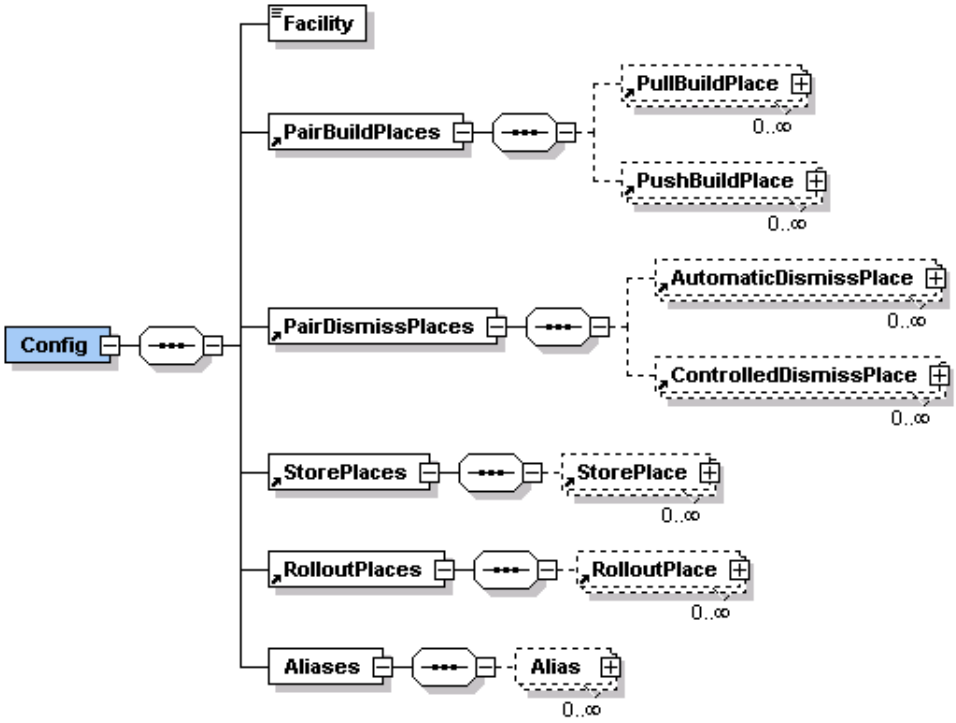


Abbildung 5.8: Schema der Konfigurationsdatei

Für das Einlesen der XML-Datei wird der Standard-SAX-Parser von Java verwendet. SAX erhielt gegenüber DOM den Vorzug, weil sich das Konzept von SAX für unsere Aufgabe, das File sequenziell einzulesen, perfekt eignet. So war die Programmierung einfach und es werden weniger Ressourcen benötigt. Abbildung 5.9 zeigt die wichtigsten Klassen, welche beim Einlesen der Konfiguration zum Einsatz kommen. Der `ConfigReader` erstellt den SAX-Parser. Der `ConfigHandler` wird vom SAX-Parser aufgerufen und übergibt dem `ConfigReader`, über das `ConfigCallback`, die Konfiguration, wenn er sie fertig eingelesen hat. Die Klasse `Configuration` beinhaltet alle in der Konfigurationsdatei enthaltenen Elemente.

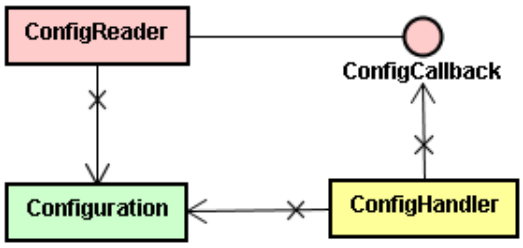


Abbildung 5.9: Konfiguration

6 Projektablauf und Entscheidungen

Dises Kapitel soll Aufschluss über den Verlauf unserer Arbeit geben. Zuerst gibt es einen Überblick über die Grobplanung, danach folgen Abschnitte über die Entwicklungsphasen, wo kurz unser Fortschritt und die wichtigsten Entscheidungen, welche getroffen wurden, beschriebem sind.

6.1 Übersicht

Die Abbildung 6.1 zeigt einen groben Zeitplan des Projekts. Der Projektverlauf orientiert sich am itarativen Vorgehen des RUP: Geplant wurden drei Implementations-Iterarionen mit jeweils einem lauffähigen Prototypen am Ende der Iteration.

6.2 Inception

In der Inception-Phase ging es uns darum, einen ersten Überblick über die Lagerthematik zu bekommen. Wir hatten einige Termine bei Stöcklin Software und erhielten eine Einführung sowie die Simulation einer Anlage, damit wir uns mit LAKOS vertraut machen konnten. In dieser Phase erstellten wir den Projektplan, sammelten Ideen für das Design und fertigten die erste GUI-Skizze an.

6.3 Elaboration

Auch die Elaboration stand noch im Zeichen der Einarbeitung. Wir besichtigten das Manor-Lager in Hochdorf. Hinzu kamen die Domainanalyse und eine detaillierte Planung des ersten Prototypen. Wir entschieden uns, diejenige Variante zu implementieren, in welcher die Telegramme jeweils ihrer Sendezeit entsprechend animiert und wie in einem Player abgespielt werden.

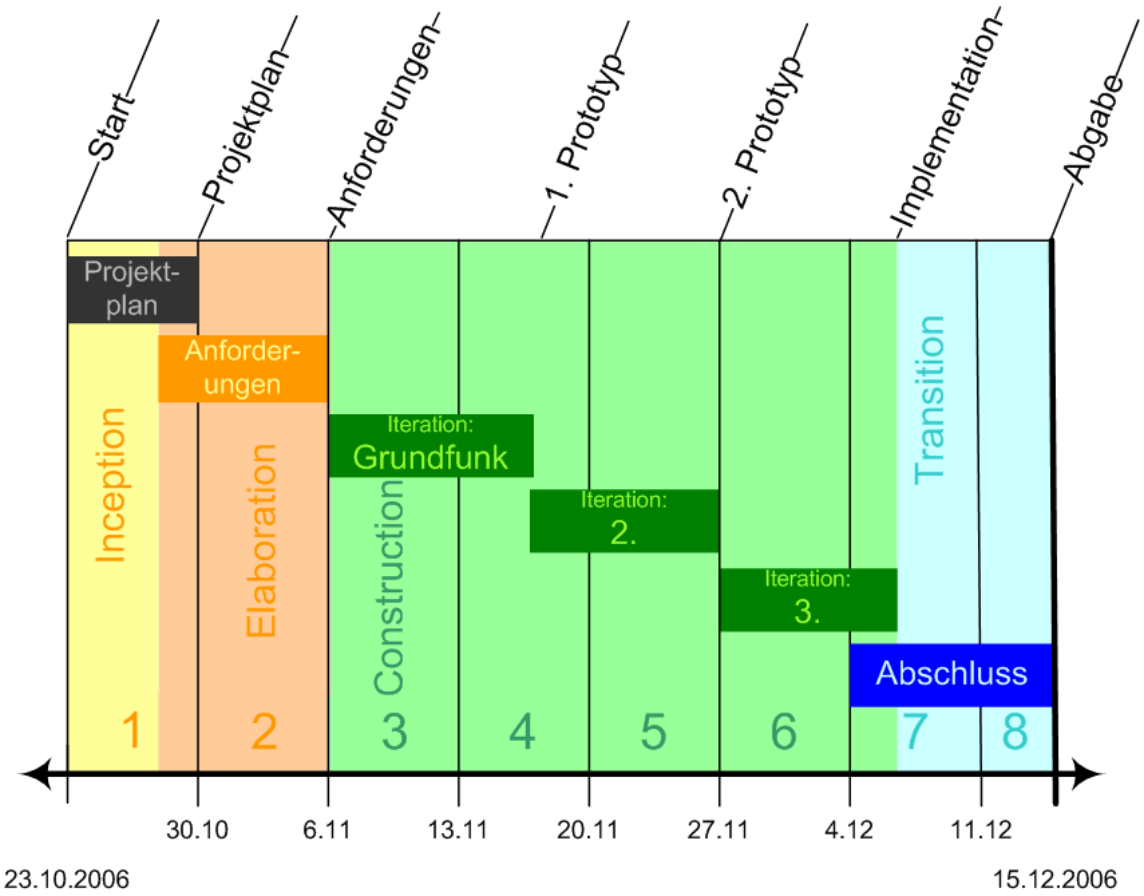


Abbildung 6.1: Projektplan

6.4 Construction

1. Iteration Ziel in der ersten Iteration war die Erstellung des GUIs, des Parsers und einem Wiedergabemechanismus ohne Filter. Dieses Ziel erreichten wir, mussten aber merken, dass unsere Darstellungsart nicht sehr praktisch ist und die Animation das Arbeiten mit dem Programm eher erschwerte, denn etwas brachte.

2. Iteration In der zweiten Iteration stellten wir unsere Darstellung um: Die Animation wurde entfernt und die Telegramme wurden auf der Zeitachse nicht mehr linear, sondern nur noch chronologisch nach der Sendereihenfolge dargestellt. Ebenfalls erstellten wir die Datenbank und schrieben den Parser so um, dass die Telegramme in die Datenbank geschrieben wurden. Dies erleichterte uns die Programmierung von Filtern. Auch die Konfigurationsdatei und das Einlesen der Konfiguration erstellten wir in der zweiten Iteration.

3. Iteration In der letzten Iteration programmierten wir die noch fehlenden Filter und erweiterten das GUI entsprechend. Grosse Änderungen bezüglich des Designs gab es nicht mehr.

6.5 Transition

Während der Transition-Phase wurde die Applikation nochmals getestet und der Code erfuhr eine letzte Überarbeitung. Ein grosser Teil der Arbeitszeit floss auch in die Dokumentation: Bestehende Dokumente wurden überarbeitet und ein Teil der Dokumentation konnte erst in dieser Phase geschrieben werden.

6.6 Zeitaufwand

In den Abbildungen 6.2 und 6.3 sind die Arbeitsstunden pro Woche bzw. Kategorie mit Soll- und Ist-Werten aufgeführt. Ein detaillierter Zeitplan ist in Kapitel 10.1 zu finden.

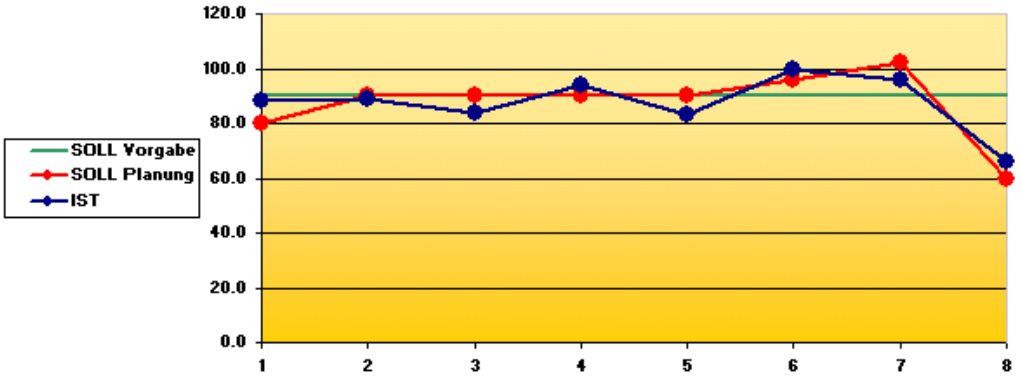


Abbildung 6.2: Arbeitsstunden pro Woche

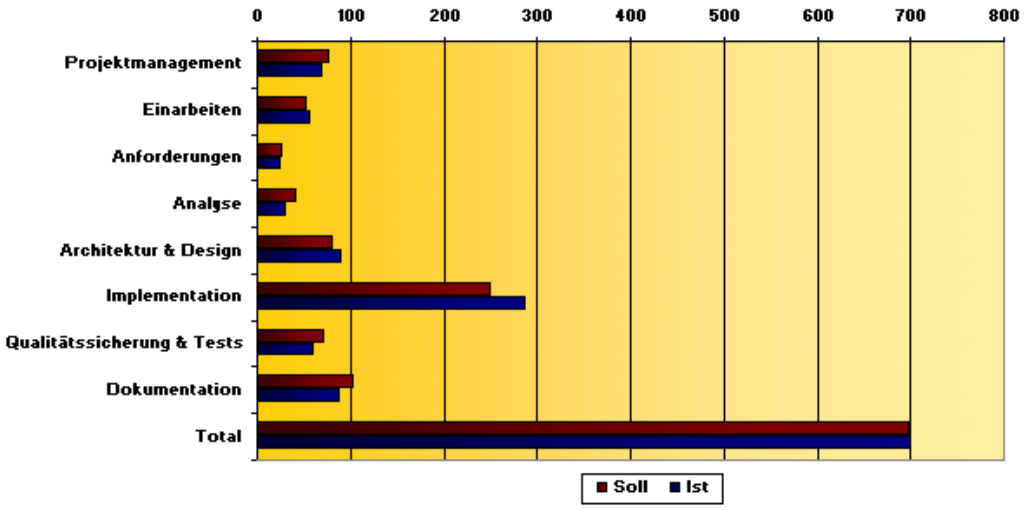


Abbildung 6.3: Arbeitsstunden pro Kategorie

6.7 Codestatistik

In der Tabelle 6.1 sind die Anzahl Codezeilen und die Anzahl der Klassen aufgeführt. Gezählt wurden sie mit dem Eclipse-Plugin Metrics.

Paket:	**Klassen:**	**Zeilen:**
ch.hsr.lfv	1	23
.pd		
.auxiliaryFunctions	3	134
.playback	5	568
.telegrams	4	354
.tests	2	546
.pe	1	14
.config	4	534
.elements	10	260
.database	6	1072
.parser	7	286
.tests	7	1137
.ui	4	1122
.date	3	146
.dialogs	2	118
.languages	1	18
.panels	2	390
.telegram	2	88
.buttons	3	140
.popummenus	3	165
Ohne Tests:	61	5432
Total:	**70**	**7115**

Tabelle 6.1: Codezeilen

7 Fazit

7.1 Erreichtes

Die erstellte Applikation erbringt die von der Firma Stöcklin Software AG geforderte Funktionalität. Die von uns realisierte Visualisierung der Logfiles ermöglicht es, Abläufe wie die Ein- oder Auslagerung eines Behälters auf Telegrammebene zu verfolgen. Damit erhält Stöcklin Sofware ein Werkzeug, welches es erlaubt, Abläufe neu entwickelter Anlagen anhand der Logfiles zu überprüfen oder auch Ungereimtheiten zu erkennen. Als weiteres Anwendungsgebiet unserer Anwendung wurde die Dokumentation erkannt; so müssen nicht mehr für alle Abläufe von Hand Sequenzdiagramme erstellt werden sondern die Abläufe werden dynamisch aus den Logfiles dargestellt.

7.2 Limitierungen

Unsere Anwendung verfügt natürlich auch über Einschränkungen. So ist es nicht möglich, die Applikation über eine Konfigurationsdatei vollständig an eine neue Anlage anzupassen. Auch wenn mit den angebotenen Konfigurationen ein paar Eigenheiten der Anlagen unterschiedlich eingestellt werden können, wird es nötig sein, für Spezialfälle Anpassungen am Source-Code vorzunehmen. Wir hoffen aber, dass dies aufgrund unseres Designs und der Dokumentation jeweils problemlos machbar ist.

7.3 Ausblick

Ein perfekter LogFileVisualizer wäre vollständig konfigurierbar und liesse sich somit per Konfigurationsdatei an jede beliebige Anlage anpassen. Optimal wäre es, wenn auch die Filter dynamisch erzeugt werden könnten, so dass je nach Anlage die benötigten Filter eingebunden würden. Diese Flexibilität könnte zum Beispiel über ein regelbasiertes System erreicht werden, welches eine Trennung der Wissensbasis von der Programmlogik ermöglicht.

Ein anderer Punkt wäre die Anbindung der Applikation an das bestehende Visualisierungssystem von Stöcklin Software. Dadurch würde es möglich, anhand der Logfiles die gleiche Darstellung zu erhalten, wie sie bei der Simulation des Lagers existiert.

II

Projektplan

Verantwortlich: Rico Steffen

Datum	Version	Änderung
25. 10. 2006	0.1	Erstellt
25. 10. 2006	0.2	Projektübersicht, Projektorganisation
25. 10. 2006	0.3	Planung
25. 10. 2006	0.4	Risikomanagement
26. 10. 2006	0.5	Arbeitspakete
27. 10. 2006	0.6	Arbeitspakete
04. 11. 2006	0.7	Risikomanagement
10. 12. 2006	0.8	Überarbeitung
13. 12. 2006	1.0	Final

8 Projektübersicht

8.1 Projektidee

Im laufenden Betrieb einer Anlage werden u.a. sogenannte SPS-Telegramme zwischen dem Materialfluss-Teil des Lagerverwaltungssystems und den maschinennahen Steuerungen (SPS, i.d.R. Siemens S7) hin und her geschickt. Dieser Telegrammverkehr wird in einem (grossen) Logfile protokolliert.

Die Projektidee wäre eine Software, die diesen Telegrammverkehr analysiert und im optimalen Fall ein «Replay» eines bestimmten Zeitabschnittes animiert darstellen kann. Zur Fehlersuche wäre es extrem hilfreich, wenn ein Software-Entwickler quasi die «Video-Aufnahme» einer real abgelaufenen Szene von einer Anlage nochmals in Slow Motion anschauen könnte.

8.2 Ziel und Zweck

Die Software soll bei der Stöcklin Software AG zum Einsatz kommen. Ziel ist es, den Entwicklern ein Tool zu bieten, welches sie bei der Analyse des Systems unterstützen kann. Damit das Tool einfach angepasst oder erweitert werden kann, ist ein sauberer Aufbau der Software und eine entsprechend gute Dokumentation notwendig. Wenn möglich sollen auch Konfigurationsmöglichkeiten im Programm selbst angeboten werden. Ein Einsatz derselben Entwicklungsplattform, die auch bei LAKOS eingesetzt wurde, soll eine spätere Integration ermöglichen.

9 Projektorganisation

9.1 Projektteam

Das Projektteam (Tabelle 9.1) besteht aus zwei Personen und wird von Daniel Keller betreut. Die Ansprechpersonen bei der Stöcklin Software AG sind in Tabelle 9.2 aufgeführt.

Mitglied	Kürzel	E-Mail
Roland Fehlmann	RF	rfehlman@hsr.ch
Rico Steffen	RS	rsteffen@hsr.ch

Tabelle 9.1: Projektteam

Ansprechperson	Zuständig	E-Mail
Tommi Lindgren	Software	t.lindgren@stoecklin.com
Tom Lukacevic	Allgemein	t.lukacevic@stoecklin.com
Roman Widmer	Software	r.widmer@stoecklin.com

Tabelle 9.2: Ansprechpersonen bei Stöcklin Software

9.2 Infrastruktur

Arbeitsrechner Jedes Teammitglied verfügt über einen von der Schule gestellten Rechner. Auf diesem sind alle benötigten Werkzeuge zur Entwicklung und Dokumentation installiert. Zusätzlich besitzt jedes Mitglied ein Notebook, welches über die gleiche Software verfügt.

Testrechner Das Projektteam besitzt einen zusätzlichen Computer. Dieser wurde bei der Stöcklin Software AG mit der nötigen Software ausgerüstet, um das Lager von Capaldo zu simulieren.

Server Für das Projekt wird ein Server benutzt, welcher von einem Teammitglied privat zur Verfügung gestellt wird. Auf dem Server ist eine Versionskontrolle (Subversion), ein Webserver und ein Wiki installiert.

Software Zum Entwickeln der Applikation wird Eclipse eingesetzt [Fou05]. UML Diagramme erstellen wir mit JUDE. Für die Dokumentation wird LaTeX [Hel02] [Koh04], JUDE und Microsoft Office verwendet. Eine detaillierte Auflistung der verwendeten Software ist im Anhang A zu finden.

Organistation / Kommunikation Das Wiki wird zur Verwaltung von Notizen, Terminen, Ideen usw. verwendet. Als Hilfsmittel zur Kommunikation werden E-Mail, Telefon und Instant Messaging eingesetzt.

9.3 Organisation

Arbeitszeit Es ist vorgesehen, von Montag bis Freitag jeweils den ganzen Tag (ca. 8 Stunden zwischen 8 und 17 Uhr), in Rapperswil zu arbeiten. Dadurch wird sichergestellt, dass die teaminterne Kommunikation optimal funktionieren kann. Zusätzlich zur Arbeit in Rapperswil sollte jedes Teammitglied noch rund 5 Stunden pro Woche in die Diplomarbeit investieren.

Aufgaben Eine Auflistung der Aufgaben ist im Detailzeitplan in Kapitel 10.1: *Zeitplan* zu finden. Im Wiki befindet sich zudem eine aktuelle ToDo-Liste, welche laufend angepasst wird.

Dokumentation Für jeden Teil der Dokumentation ist ein Projektmitglied verantwortlich. Der Verantwortliche hat dafür zu sorgen, dass der Teil vollständig und richtig ist. Änderungen der Dokumentteile werden auf dem Deckblatt der jeweiligen Teile kurz aufgeführt.

QM Als qualitätssichernde Massnahmen werden jeweils zur Fertigstellung der Prototypen teaminterne Codereviews durchgeführt. Mindestens ein Condereview soll zusätzlich mit dem Betreuer stattfinden. Getestet wird mit Unit-Tests sowie der Durchführung von Usability-Tests. Genauere Infos zum Testen sind im Dokumentteil *Test* zu finden – ab Kapitel 21.

Meetings Zur Überprüfung des Arbeitsfortschrittes und der Klärung allfälliger Fragen findet ein wöchentliches Meeting mit dem Betreuer statt. Zu Beginn der Arbeit ist eine intentensivere Betreuung vorgesehen. Bei Problemen während der Arbeit sind zusätzliche Termine möglich (auch kurzfristig).

Stöcklin Software AG Die Kommunikation mit der Stöcklin Software AG soll hauptsächlich per Mail erfolgen. Dringende Probleme können auch per Telefon oder vor Ort gelöst werden. Die Ansprechpersonen sind in Tabelle 9.2 aufgeführt.

10 Planung

10.1 Zeitplan

Die Abbildung 10.1 zeigt einen groben Zeitplan des Projekts.

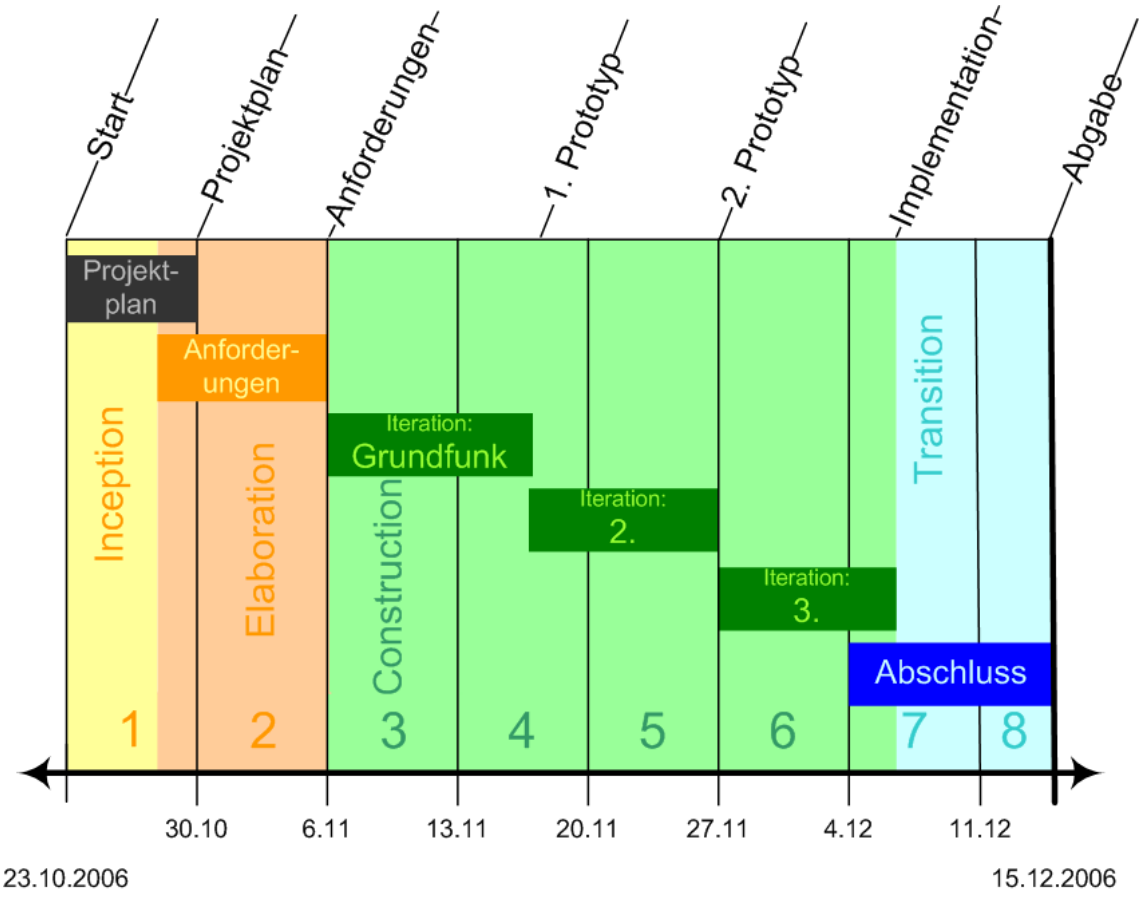


Abbildung 10.1: Projektplan

Auf den folgenden Seiten ist der ausführliche Zeitplan zu sehen. In diesem sind sind die Arbeitspakete sowie Soll- und Ist-Zeiten eingetragen.

10.2 Projektphasen

Die Projektphasen entsprechen den Phasen von RUP[1] und werden nachfolgend kurz erläutert.

Inception In der Inception-Phase werden erste Gedanken über die zu erstellende Software gemacht und ein grober Projektplan wird erstellt. Hinzu kommt das Einrichten der Arbeitsumgebung sowie das Einarbeiten in das Lagerverwaltungs- und Kommissioniersystem LAKOS.

Elaboration Hier werden die Anforderungen spezifiziert und eine erste Analyse sowie ein Architektur-Entwurf erstellt.

Construction Die Construction-Phase beinhaltet die eigentliche Implementation der Software. In unserem Projekt sind drei Iterationen vorgesehen, in welchen jeweils ein Analyse-, Design-, Implementierungs-, und Test-Durchgang erfolgt. Zusätzlich werden die neuen Teile dokumentiert. Nach jeder Iteration wir ein lauffähiger Prototyp erstellt.

Transition In dieser Phase werden letzte Tests durchgeführt, die Dokumentation wird fertiggestellt und das Projekt für die Abgabe vorbereitet.

10.3 Meilensteine

Die jeweiligen Daten der Meilensteine sind dem Abschnitt 10.1 zu entnehmen.

Projektplan Am Ende dieses Meilensteines ist der Projektplan fertiggestellt. Der Zeitplan wird während des ganzen Projektes laufend angepasst und die Arbeitspakete der Iterationen werden dem Projektverlauf entsprechend festgelegt.

Anforderungen Zu diesem Zeitpunkt muss die Anforderungsspezifikation fertiggestellt sein. In dieser wird festgelegt, was im Rahmen der Diplomarbeit umgesetzt wird.

1. Prototyp Eine erste lauffähige Version der Software wird erstellt. Diese verfügt über einen reduzierten Funktionsumfang, zeigt aber wie das Produkt aussehen könnte. Eine detaillierte Liste der zu implementierenden Funktionen aller Prototypen ist in Kapitel 15: *Prototypen* zu finden.

[1] Rational Unified Process

2. Prototyp	Ein zweiter Prototyp der Software wird erstellt. Dieser besitzt eine erweiterte Funktionalität und zeigt, wie das fertige Produkt aussehen wird.
Implementation	Die Applikation muss, bis auf allfällige Fehlerkorrekturen, zu diesem Meilenstein fertig implementiert sein.
Abgabe	Die Arbeit am Projekt ist abgeschlossen, die Applikation ist lauffähig und alle Dokumente sind fertiggestellt.

11 Risiko Analyse

Die Risiken werden nachfolgend aufgelistet und erläutert. In der Tabelle 11.1 werden die Eintrittswahrscheinlichkeiten, Kosten und Massnahmen aufgeführt.

- **Definierte Anforderungen zu hoch**
  Unterschätzen der Komplexität.
- **Probleme bei der Zusammenarbeit mit Stöcklin**
  Durch zu seltene Kommunikation mit Stöcklin entsteht ein unbrauchbares Produkt.
- **Mangelhafte Unterstützung durch Stöcklin**
  Die Kommunikation mit den Mitarbeitern von Stöcklin funktioniert nicht; unsere Probleme, Anliegen werden ignoriert.
- **Arbeitsunfähigkeit eines Teammitglieds**
  Ein Teammitglied wird infolge eines Unfalles oder einer Krankheit arbeitsunfähig oder beeinträchtigt.
- **Ausfall des Testsystems**
  Ausfall des Rechners, auf welchem das Testsystem installiert ist.
- **Computerausfall**
  Ausfall eines Arbeitsrechners.
- **Datenverlust**
  Ausfall des Servers.
- **Ausstieg eines Teammitglieds**
  Ein Teammitglied verlässt das Team.

[1] Eintrittswahrscheinlichkeit
[2] Kosten in Stunden
[3] Faktor = Eintrittswahrscheinlichkeit × Kosten in Stunden

Risiko	**p**[1]	**h**[2]	$\times$[3]	**Vorbeugung**	**Massnahmen**	**Frühwarnsignale**
Anforderungen zu hoch	0.1	40	4	Gutes Einarbeiten und Analyse der Anforderungs-spezifikation mit dem Betreuer.	Mit dem Betreuer absprechen. Abstriche bei den Anforderungen	wenig oder keine Fortschritte mehr
Probleme bei der Zusammenarbeit	0.1	20	2	Regelmässige Kommunikation, Demonstration der Prototypen	Mehr Absprechen mit den zuständingen Personen	keine
Mangelhafte Unterstützung	0.1	15	1.5	Professionelles Verhalten	Geduld und viel eigen Initative	Lange Antwortzeiten auf Fragen und ungenaue Informationen
Arbeitsunfähigkeit	0.05	20	1	Redundantes Wissen	Restrukturierung der Arbeitseinteilung, ev. Abstriche beim Funktionsumfang.	keine
Ausfall des Testsystems	0.05	8	0.4	keine	Neuinstallation	keine
Computerausfall	0.05	8	0.4	Zweit-PC und Backup auf SVN-Server	Neuinstalltion	keine
Datenverlust	0.05	8	0.4	Tägliches Backup	Aufarbeiten der verlorenen Arbeit	keine
Ausstieg	0.01	20	0.1	Redundantes Wissen	(wie bei *Arbeitsunfähigkeit*)	Unstimmigkeiten im Team
Total:			9.8		Prozent mehr Zeit berechnen	

Tabelle 11.1: Risiken

III

Anforderungsspezifikation

Verantwortlich: Roland Fehlmann

Datum	Version	Änderung
25. 10. 2006	0.1	Erstellt
26. 10. 2006	0.2	Diverses
26. 10. 2006	0.3	Nichtfunktionale Anforderungen
26. 10. 2006	0.4	User Interface
27. 10. 2006	0.5	Use Cases
30. 10. 2006	0.6	UC Filter
30. 10. 2006	0.7	1. Prototyp
15. 11. 2006	0.8	2. Prototyp
13. 12. 2006	1.0	Final

12 Allgemeine Beschreibung

12.1 Zweck

Die Visualisierung des Meldungsflusses dient hauptsächlich als Unterstützung für den Entwickler. Die Möglichkeit, Logfiles nicht nur textuell betrachten zu können, sondern visualisiert, mit der Funktion nach verschiedenen Kriterien filtern zu können, soll die Analysearbeit erleichtern. Auch zur Einarbeitung in ein System kann die Software hilfreich sein.

12.2 Funktion

Die Software soll Logfiles visualisieren. Das heisst, die in den Logfiles protokollierten Meldungen und Telegramme werden in einer Art und Weise dargestellt, die es erlaubt, den Ablauf nachvollziehen zu können. Dazu werden die Protokolle von zwei verschiedenen Schnittstellen verwendet und nach Möglichkeit miteinander in Verbindung gebracht.

12.3 Benutzerrollen

Es werden grundsätzlich zwei verschiedene Benutzerrollen unterschieden. Diese werden in den folgenden Abschnitten genauer beschrieben.

Entwickler Unter einem Entwickler wird eine Person verstanden, welche an der Softwareentwicklung einer Logistikanlage beteiligt ist. Somit sollte diese über den Ablauf im System bescheid wissen.

Der Entwickler sollte die Software, nach einer kleinen Einführung mit Hilfe des Benutzerhandbuches, ohne Probleme bedienen können.

Supporter Der Supporter ist eine Person, welche für die verschiedenen Anlagen die im Betrieb sind, Hilfe leistet.

12.4 Einschränkungen

Da die Steuerungssoftware der Logistikanlagen sehr komlex ist und die Zusammenhänge der MF-Telegramme mit den SPS-Telegrammen nur wenig dokumentiert sind, dürfte es nicht möglich sein, für alle Abläufe Filter zu implementieren. Deshalb wäre eine einfache Erweiterbarkeit wünschenswert.

13 Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen lehnen sich am Modell FURPS+ von Larman [Lar04] an, welches auch im ISO-Standard 9126 normiert ist.

13.1 Usability

Erlernbarkeit LogFileVisualizer soll intuitiv zu bedienen sein. Ein Benutzer, welcher LAKOS bedienen kann, sollte sich rasch in der Software zurecht finden.

Bedienbarkeit Zur sinnvollen Bedienung von LogFileVisualizer werden grundlegende Kenntnisse der Telegramm- und Meldungs-Typen vorausgesetzt. Die Arbeit mit LogFileVisualizer soll sich so einfach gestalten, dass sich der Benutzer auf die eigentliche Aufgabe konzentrieren kann.

13.2 Reliability

Robustheit Die Applikation darf nicht abstürzen, wenn fehlerhafte oder sehr grosse Logfiles eingelesen werden sollen.

13.3 Performance

Speicherbedarf Auch bei sehr grossen Logfiles (bis 200MB) darf die Applikation nicht zu viel Speicher belegen[1].

Visualisierung Die Animationen sollen auf einem aktuellen PC-System[2] flüssig laufen.

[1] Maximal 300MB zur Laufzeit
[2] Pentium® 4 oder Ähnliches mit mindestens 512MB RAM

13.4 Supportability

Integrations-möglichkeit Die Software sollte sich einfach in das bestehende LAKOS integrieren lassen.

Anpassbarkeit Bei Änderungen des Logfile-Formats sollte das Programm schnell an die Strukturen angepasst werden können.

13.5 Implementation requirements

Sprache Das Programm soll mit Java geschrieben werden, das GUI wird mit Swing erstellt. Die Software wird vorläufig nicht in das LAKOS der Firma Stöcklin integriert, durch den Einsatz von Java und Swing sollte eine spätere Integration jedoch kein Problem darstellen.

14 Funktionale Anforderungen

14.1 Use Case Diagramm

Unser Programm soll folgenden Use Cases abdecken:

Logfile analysieren Der Benutzer möchte den Inhalt eines Logfiles analysieren und Zusammenhänge zwischen Telegrammen des SPS-Logs und solchen des WMS-Logs herausfinden.

Die Abbildung 14.1 zeigt das Use Case Diagramm.

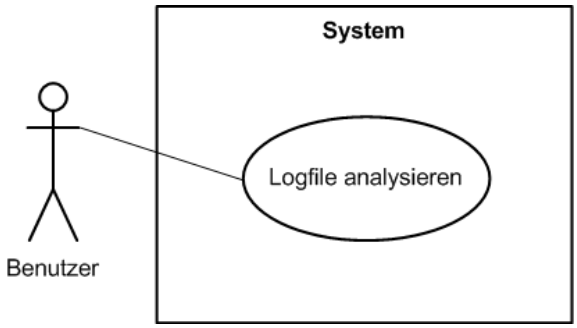


Abbildung 14.1: Use Case Diagramm

14.2 Use Case: Logfile analysieren

Die neu entwickelte Anlage funktioniert noch nicht wunschgemäss. Immer, wenn von einem bestimmten Platz aus eine Palette eingelagert werden soll, fährt sie wieder aus dem Lager heraus. Der Entwickler Marco Minder erhält den Auftrag, den Fehler zu lokalisieren und ihn zu beheben. Dazu reproduziert er den Fehler in der Simulation der Anlage und versucht anschliessend, anhand der Logfiles, die Abläufe nachzuvollziehen und herauszufinden, welche Telegramme falsch gesendet wurden. Dazu verwendet er unsere Applikation:

1. Der Entwickler lädt die beiden Logfiles der Schnittstellen zwischen LVS und MF sowie zwischen MF und SPS.

2. Das System zeigt an, welche Logfiles der Benutzer geladen hat.
3. [Der Entwickler lädt die Konfigurationsdatei.]
4. Der Entwickler definiert die Filter, welche er anwenden möchte und aktiviert die Anzeige.
5. Das System visualisiert die gefundenen Telegramme.

14.3 Funktionsliste

Nachfolgend werden die einzelnen Funktionen dargestellt, welche mit dem Programm ausgeführt werden können.

wms.log einlesen

Preconditions LogFileVisualizer wurde gestartet.

Postconditions Das WMS-Logfile wurde eingelesen.

Basic flow
1. Der Benutzer sucht ein WMS-Log, welches eingelesen werden soll.
2. Das Logfile wird eingelesen.

sps.log einlesen

Preconditions LogFileVisualizer wurde gestartet.

Postconditions Das SPS-Logfile wurde eingelesen.

Basic flow
1. Der Benutzer sucht ein SPS-Log, welches eingelesen werden soll.
2. Das Logfile wird eingelesen.

Konfigurations-Datei einlesen

Preconditions LogFileVisualizer wurde gestartet.

Postconditions Die Konfigurations-Datei wurde eingelesen.

Basic flow
1. Der Benutzer sucht die einzulesende Konfigurations-Datei.
2. Die Konfiguration wurde eingelesen.

Alle Telegramme darstellen

Preconditions	Die Logfiles wurden eingelesen, eventuell wurden Filter gesetzt.
Postconditions	Alle Telegramme werden dargestellt.
Basic flow	1. Benutzer wählt «Anzeigen». 2. Die aus den Logfiles geladenen Telegramme werden dargestellt.

Gefilterte Telegramme darstellen

Preconditions	Die Logfiles wurden eingelesen, eventuell wurden Filter gesetzt.
Postconditions	Alle passenden Telegramme werden dargestellt.
Basic flow	1. Benutzer wählt «Anzeigen». 2. Die aus den Logfiles geladenen und gefilterten Telegramme werden dargestellt.

Anzeige löschen

Preconditions	Die Logfiles wurden eingelesen, Telegramme werden dargestellt.
Postconditions	Es werden keine Telegramme mehr dargestellt.
Basic flow	1. Benutzer wählt «Anzeige löschen». 2. Die Anzeige wird gelöscht.

Telegrammdetails anzeigen

Preconditions	Die Logfiles wurden eingelesen und mindestens ein Telegramm wird dargestellt.
Postconditions	Die Telegrammdetails wurden angeschaut.
Basic flow	1. Benutzer wählt das gewünschte Telegramm aus. 2. Detail-Informationen zum Telegramm werden angezeigt.

Intervall der angezeigten Zeitlinien ändern

Preconditions Die Logfiles wurden eingelesen.

Postconditions Die Anzahl Telegramme pro Zeitlinie ist anders.

Basic flow

1. Benutzer gibt an, nach wie vielen Telegrammen eine Zeitlinie angezeigt wird.
2. Beim nächsten Darstellen der Telegramme wird die gewählte Anzahl Zeitlinien angezeigt.

Anzahl-Filter anwenden

Preconditions Die Logfiles wurden eingelesen.

Postconditions Es werden maximal so viele Teleramme angezeigt, wie im Filter definiert sind. Dabei werden diejenigen Telegramme abgeschnitten, welche zu späterer Zeit gesendet wurden.

Basic flow

1. Benutzer aktiviert den Startdatum- oder/und Enddatum-Filter.
2. Benutzer legt Daten fest.
3. Der Filter wird angewandt.

RegExp-Filter anwenden

Preconditions Die Logfiles wurden eingelesen.

Postconditions Nur die Telegramme, welche den Regulären Ausdruck enthalten, werden angezeigt.

Basic flow

1. Benutzer aktiviert den Filter.
2. Benutzer bestimmt den Regulären Ausdruck.
3. Der Filter wird angewandt.

Datum-Filter anwenden

Preconditions Die Logfiles wurden eingelesen.

Postconditions Nur die Telegramme, welche nach dem Von-Datum und vor dem Bis-Datum gesendet wurden, werden angezeigt.

Basic flow

1. Benutzer aktiviert den Startdatum- oder/und Enddatum-Filter.
2. Benutzer legt Daten fest.
3. Der Filter wird angewandt.

Platz-Filter anwenden

Preconditions Die Logfiles und das anlagenspezifische Config-File wurde eingelesen.

Postconditions Es werden alle Telegramme, welche sich auf die angegebenen Plätze beziehen, angezeigt.

Basic flow

1. Benutzer aktiviert den Platz-Filter.
2. Benutzer gibt die gewünschten Plätze an.
3. Der Filter wird angewandt.

Telegrammtyp-Filter anwenden

Preconditions Die Logfiles wurden eingelesen.

Postconditions Alle Telegramme von den angegebenen Typen werden angezeigt.

Basic flow

1. Benutzer aktiviert den Telegrammtyp-Filter.
2. Benutzer gibt die gewünschten Telegrammtypen an.
3. Der Filter wird angewandt.

Einschaltgruppen-Filter anwenden

Preconditions Die Logfiles wurden eingelesen.

Postconditions Es werden alle Telegramme, welche sich auf die angegebene Einschaltgruppe beziehen, angezeigt.

Basic flow

1. Benutzer aktiviert den Einschaltgruppen-Filter.
2. Benutzer gibt die gewünschte Einschaltgruppe an.
3. Der Filter wird angewandt.

Carrier-Filter anwenden

Preconditions Die Logfiles und das anlagenspezifische Config-File wurde eingelesen.

Postconditions Alle Telegramme, welche sich auf einen bestimmten Carrier beziehen, werden angezeigt.

Basic flow
1. Benutzer aktiviert den Carrier-Filter.
2. Benutzer gibt die CarrierId des zu trackenden Carriers an.
3. Der Filter wird angewandt.

Mehrere Filter kombiniert anwenden

Preconditions Die Postconditions der anzuwendenden Filter sind erfüllt.

Postconditions Die Filter werden UND-verknüpft. Es werden nur diejenigen Telegramme dargestellt, welche von allen angewendeten Filtern selektiert werden.

Basic flow
1. Benutzer selektiert mehrere Filter und konfiguriert sie entsprechend.
2. Alle selektierten Filter werden angewendet.

15 Prototypen

15.1 Prototyp: Grundfunktionen

Dieser Prototyp soll nach der ersten Iteration (Kapitel 10.1: *Zeitplan*) fertiggestellt sein. Folgende Funktionen (siehe Kapitel 14.3) sollen umgesetzt werden, die Bedienung soll bereits über eine grafische Benutzeroberfäche erfolgen:

Funktionalität	**vollständig**	**teilweise**	**nicht**
wms.log einlesen	×		
sps.log einlesen	×		
Konfigurations-Datei einlesen			×
Alle Telegramme darstellen	×		
Gefilterte Telegramme darstellen		×	
Anzeige löschen	×		
Telegrammdetails anzeigen	×		
Intervall der angezeigten Zeitlinien ändern	×		
Anzahl-Filter anwenden	×		
RegExp-Filter anwenden			×
Datum-Filter anwenden			×
Platz-Filter anwenden			×
Telegrammtyp-Filter anwenden			×
Einschaltgruppen-Filter anwenden			×
Carrier-Filter anwenden			×
Mehrere Filter kombiniert anwenden			×

Tabelle 15.1: Implementierte Funktionen im 1. Prototyp

15.2 Prototyp: Erweiterte Funktionalität

Das ist der Prototyp nach der zweiten Iteration (Kapitel 10.1: *Zeitplan*). Die Funktionen in der Tabelle 15.2 werden umgesetzt.

Funktionalität	**vollständig**	**teilweise**	**nicht**
wms.log einlesen	×		
sps.log einlesen	×		
Konfigurations-Datei einlesen		×	
Alle Telegramme darstellen	×		
Gefilterte Telegramme darstellen	×		
Anzeige löschen	×		
Telegrammdetails anzeigen	×		
Intervall der angezeigten Zeitlinien ändern	×		
Anzahl-Filter anwenden	×		
RegExp-Filter anwenden	×		
Datum-Filter anwenden	×		
Platz-Filter anwenden			×
Telegrammtyp-Filter anwenden			×
Einschaltgruppen-Filter anwenden	×		
Carrier-Filter anwenden		×	
Mehrere Filter kombiniert anwenden			×

Tabelle 15.2: Implementierte Funktionen im 2. Prototyp

15.3 Prototyp: Final

Dieser Prototyp soll am Ende der Construction-Phase (Kapitel 10.1: *Zeitplan*) fertig sein. Der Use Case (siehe Abschnitt 14.2) soll vollständig umgesetzt sein, alle Funktionen sind implementiert.

IV

Analyse

Verantwortlich: Roland Fehlmann

Datum	Version	Änderung
27. 10. 2006	0.1	Erstellt
30. 10. 2006	0.2	Domain Model
30. 10. 2006	0.3	LAKOS
30. 11. 2006	0.4	Telegramme
13. 12. 2006	1.0	Final

16 LAKOS Systemarchitektur

Dieses Kapitel (Bild, Legende und Texte) wurde aus der Stöcklin-Dokumentation (LAKOSKonzepteV4.1) übernommen .

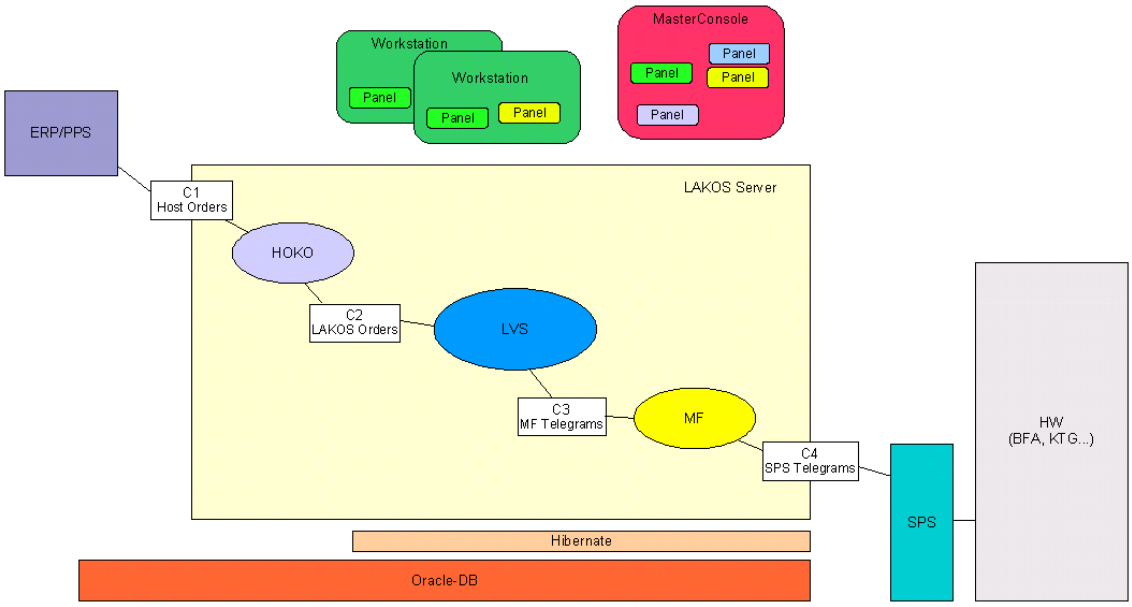


Abbildung 16.1: Systemübersicht LAKOS V4.1

ERP/PPS	**Externes System, z.B. SAP**
HOKO	*konfigurierbare Schnittstelle für Aufträge «von oben»*
LVS	*Verwaltung von Aufträgen + Material + Fächern + Paletten*
MF	*Ansteuerung SPS + Verwaltung Paletten*
SPS	*Programmierbare Steuerung (PLC, z.B. Siemens S7)*
HW	*Förderanlage, Sensoren, Motoren, KTG/RBG*
C1, C2, C3, C4	*Kommunikations-Schnittstellen*
Workstation	*Arbeitsstation für Ein-/Auslagerungen*
MasterConsole	*Überwachung, Steuerung und Korrekturen*
Panel	*Anzeige-Komponente (Maske/Tabelle/Bild)*

16.1 Architektur

Figur 16.1 zeigt eine Übersicht über alle LAKOS-Komponenten und deren Interaktion mit den vor- und nachgelagerten Systemen. In der Regel ist LAKOS an ein ERP (Enterprise Resource Planning System, z.B. SAP) oder an ein Produktionsplanungssystem angebunden (Kasten ERP/PPS links oben).

Ebenso üblich ist es, dass LAKOS zusammen mit automatischen Lagern und Förderanlagen im Einsatz ist (SPS und HW rechts).

Es ist allerdings möglich, LAKOS sowohl ohne ERP als auch ohne automatische Lager-Komponenten (manuelles Lager) zu betreiben.

16.2 Schnittstellen

In dieser Systemübersicht spielen die Schnittstellen C1 bis C4 eine wichtige Rolle. Sie sind für das saubere Zusammenspiel der Komponenten verantwortlich.

Schnittstelle C1: Host Orders

Wie ein übergeordnetes Kunden-System (z.B. SAP) die Befehle und Daten an LAKOS übergibt, ist in jedem Fall zusammen mit dem Kunden zu bestimmen. Diese Schnittstelle ist weitgehend ohne Programmierung konfigurierbar. Es können beispielsweise SAP TORDs in der Form von IDOCs übermittelt werden - die Feldinhalte sind auf LAKOS-Seite konfigurierbar, da jedes SAP anderen Konventionen für die Feldwerte folgt. Es ist auch möglich, Daten und Befehle in der Form von Kalkulations-Tabellen als CSV Dateien (Comma Separated Values) zu übermitteln. Als dritte Variante kann ein ERP-System die Daten in eine zu definierende Oracle-DB-Tabelle ablegen, welche dann von LAKOS-Seite gelesen und interpretiert wird.

Egal, in welcher Form die Daten und Befehle vom Kunden-System übermittelt werden, sie werden von der LAKOS HOKO in die interne Form der LAKOS Orders (Schnittstelle C2) übersetzt und so an das LVS übermittelt.

Schnittstelle C2: LAKOS Orders

Dies ist die standardisierte Schnittstelle für die Befehle und Daten, die zwischen dem LVS-Kern und dem jeweiligen Kunden-System hin und her übertragen werden.

Hin: Ein- und Auslageraufträge, Artikel-Stammdaten, Inventur-Anstoss. Her: Quittierungen für die erteilten Aufträge.

Schnittstelle C3: MF Telegramme

Die konkreten Fahrbefehle werden vom LVS an den MF in der Form von MF Telegrammen übertragen. Ebenso meldet sich der MF mit Statusmeldungen und Quittierungen in der Form von MF Telegrammen beim LVS zurück.

Schnittstelle C4: SPS Telegramme

Die Lager-Hardware wird von LAKOS nicht direkt angesteuert, es werden Telegramme an SPSen geschickt. Bei jeder Anlage müssen die Telegramme in Zusammenarbeit mit Stöcklin neu festgelegt werden (Form, Inhalte, wann wird welches Telegramm geschickt bzw. erwartet).

Beobachtung der Schnittstellen

Die hin und her geschickten Daten von jeder der vier Schnittstellen werden in Log-Dateien geschrieben und können so beobachtet und z.B. für die Fehlersuche auch im Nachhinein angeschaut werden.

Die ersten zwei Schnittstellen (C1 und C2) sind als Auftrags-Warteschlangen ausgebildet. Die anstehenden und die zur Zeit sich in Verarbeitung befindlichen Befehle/Daten können in der Master Konsole angeschaut und im Fehler-/Notfall sogar interaktiv verändert werden.

In unserer Arbeit werden die Logfiles der Schnittstellen C3 und C4 verwendet.

17 Domain Model

In der Abbildunge 17.1 ist das Domain Model ersichtlich. Im folgenden Kapitel werden die einzelnen Klassen kurz beschrieben. Die Beschreibung ist nach Farben sortiert.

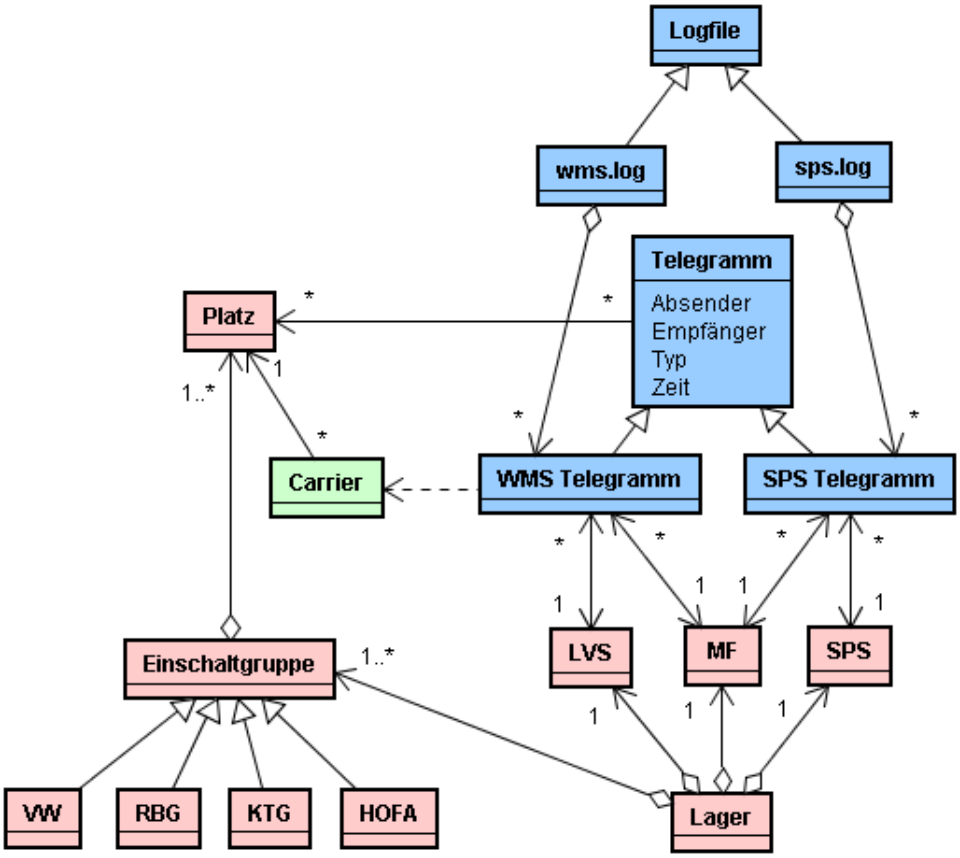


Abbildung 17.1: Domain Model

Blau

Logfile Repräsentiert ein Telegrammprotokoll.

sps.log Das Logfile welches alle Telegramme zwischen MF und SPS enthält.

wms.log	Das Logfile, welches alle Telegramme zwischen LVS und MF enthält.
Telegramm	Eine Nachricht, welche von einem System mit einem Anderen ausgetauscht wird.
SPS Telegramm	Eine Nachricht, welche zwischen MF und SPS ausgetauscht wird.
WMS Telegramm	Eine Nachricht, welche zwischen LVS und MF ausgetauscht wird.

Grün

Carrier	Ein Behälter im Lagerverwaltungssystem.

Rot

Lager	Eine komplette Lager-Anlage.
LVS	Das Lagerverwaltungssystem, welches den Materialfluss steuert.
MF	Der Materialfluss. Verantwortlich für die Bewegung der Behälter auf der Anlage.
SPS	Die speicherprogrammierbare Steuerung. Sie steuert die Hardware. Die Kardinalität wurde auf Eins gesetzt, da es um das Konzept der SPS geht, nicht um einzelne Geräte.
Einschaltgruppe	Eine logische Gruppierung von Lagerelementen.
VW	Ein Verschiebewagen.
RBG	Ein Regalbediengerät.
KTG	Ein Kleinteilgerät.
HOFA	Eine Horizontalförderanlage.
Platz	Ein Platz in einem Lager, an welchem Behälter stehen können.

18 Telegramme

Dieses Kapitel beschreibt die Telegramme, welche zwischen den einzelnen Systemen ausgetauscht werden. Dabei wird zuerst der grundlegende Aufbau beschrieben, anschliessend werden die einzelnen Telegrammtypen und ihre Funktion genauer erläutert.

Die Telegramme werden grundlegend in zwei Kategorien unterteilt: Die SPS-Telegramme, welche zwischen SPS und MF ausgetauscht werden (siehe Kapitel 16: *Systemarchitektur, Schnittstelle C4*) und die MF-Telegramme, welche zwischen MF und LVS ausgetauscht werden. Die SPS-Telegramme werden standardmässig im File sps.log geloggt, die MF-Telegramme in der Datei wms.log, weswegen sie von uns auch als WMS-Telegramme bezeichnet wurden. Die Begriffe MF-Telegramm und WMS-Telegramm werden also synonym verwendet.

Grundlage der nachfolgenden Beschreibungen bildet die Dokumentation von Stöcklin. Es werden jedoch nicht die kompletten Inhalte wiedergeben, sondern nur die für uns relevanten Teile. Partiell sind auch Informationen aus unseren Beobachtungen von realen Logfiles aus dem Betrieb der Anlagen eingeflossen.

18.1 SPS-Telegramme

SPS-Telegramme haben alle den gleichen Aufbau. Sie verfügen über einen 14 Byte langen Haeder und 76 Byte Nutzdaten. Der Header besteht aus den in Tabelle 18.1 dargestellten Felder, die Felder der Nutzdaten sind in Tabelle 18.2 aufgelistet. Nicht bei allen Telegrammen werden alle Felder verwendet. Sind Felder bei einem Telegrammtyp überflüssig oder nicht gesetzt, kommt für jedes Zeichen als Platzhalter ein Asterisk (*) zum Einsatz.

Feld	Länge in Byte	Beispiel
Kennungsfeld	2	M*
Telegrammnummer	2	01
Fehlerfeld	2	**
Telegrammquelle	4	MF01
Telegrammziel	4	PL01

Tabelle 18.1: Stöcklin-Header

Feld	Länge in Byte	Beispiel
Zone	4	0001
Von-Platz	4	1203
Nach-Platz	4	1205
Code	4	0003
PK1	4	****
PK2	4	****
Auslöser-Kenn	2	BR
Von-Gasse-Koordinate	3	001
Von-X-Koordinate	3	009
Von-Y-Koordinate	3	014
Von-Z-Koordinate	3	201
Nach-Gasse-Koordinate	3	001
Nach-X-Koordinate	3	999
Nach-Y-Koordinate	3	999
Nach-Z-Koordinate	3	411
Info1	6	******
Info2	20	*...*

Tabelle 18.2: Anwender-Prozessdaten

Wichtige SPS-Typen

Tabelle 18.3 listet alle Telegrammtypen auf, welche für das Verständnis der Abläufe unserer Software wichtig sind.

Typ	Beschreibung	Relevant	Richtung
Z*	HOFA Transportauftrag. Initiiert eine Bewegung von einem Platz zum anderen.	Von-Platz Nach-Platz Code	MFR→SPS
M*	Rückmeldung Materialflussmeldung. Folgt auf jede Palettenverschiebung sowie bei einem Wechsel des Belegungszustandes eines Aufgabe-, Abnahmeplatzes.	Von-Platz Nach-Platz Code	SPS→MFR
ZA	Fahrauftrag Achse. Löst z.B. eine Bewegung von einem VW aus.	Von-Platz Nach-Platz	MFR→SPS
MA	Rückmeldung Materialflussmeldung. Wird nach jeder Achsenbewegung, welche aufgrund eines ZA Telegramms erfolgt, gesendet.	Von-Platz Nach-Platz	SPS→MFR
ZI	HOFA Ziel ID. Legt das Ziel einer ID fest.	Von-Platz Nach-Platz	MFR→SPS
MI	Rückmeldung, welche auf eine Bewegung, die durch ein ZI ausgelöst wird, folgt.	Von-Platz Nach-Platz	SPS→MFR
BR	RBG Transportauftrag, welcher Ein- und Auslagerungen definiert.	alle Koordinaten	MFR→SPS
ED	Rückmeldung Transportauftrag beendet. Wird vom RBG erzeugt, wenn ein Transportauftrag ganz oder teilweise abgearbeitet wurde.	alle Koordinaten	SPS→MFR

Tabelle 18.3: Wichtige SPS-Telegrammtypen

18.2 WMS-Telegramme

Der Aufbau der WMS-Telegramme ist sehr telegrammspezifisch. Alle Telegramme verfügen über einen gemeinsamen Header, dessen Felder in Tabelle 18.4 beschrieben sind. Ebenfalls in dieser Tabelle befindet sich das Feld UId, welches zwar nicht

zum Header gehört, aber auch in allen WMS-Telegrammen vorhanden ist. Weitere gemeinsame Felder existieren nicht, weswegen die relevanten Felder jeweils beim Beschrieb der einzelnen Telegrammtypen erwähnt werden.

Feld	Länge in Byte	Bemerkung
Identification	6	immer SLD.CH
Direction		UP oder DOWN
UId	10	eindeutige Nummer

Tabelle 18.4: WMS-Felder

Wichtige WMS-Typen

Tabelle 18.5 listet die wichtigsten WMS-Telegrammtypen auf.

Typ	Beschreibung	Relevant	Richtung
TRA	Transportauftrag vom LVS an den MF.	UId CarrierId From	LVS→MFR
LOC	Location Status. Teilt dem LVS mit, dass ein Carrier an Platz xy steht.	UId	MFR→LVS
MLO	Multi Location Request. Beinhalet einen Array von LOC's und fordert mehrere TRA's (ein MLA) an.	UId's der LOC's	MFR→LVS
MLA	Multi Location Answer. Beinhaltet einen Array von TRA's, welche danach ohne Unterbrechung ausgeführt werden können.	UId's der TRA's	LVS→MFR
SUM	Setup Message. Wird gesendet, nachdem ein Carrier aufgesetzt wurde.	Location	MFR→LVS
CMX	Command Execution. Dient dazu, dem MFR Befehle vom LVS zu schicken. Zum Beispiel ein Kaltstart-Kommando.	TransportType TargetObject	LVS→MFR

Tabelle 18.5: Wichtige WMS-Telegrammtypen

18.3 Abläufe

Die verschiedenen Abläufe können je nach Anlage unterschiedlich ablaufen. In diesem Abschnitt werden als Beispiel ein paar Abläufe der Anlage Capaldo beschrieben.

Kaltstart

Bei einem Kaltstart (Beispiel HOFA) kommt zuerst vom LVS ein CMX an den MFR mit dem Befehl «Kaltstart». Danach folgt eine Kontrollmeldung des MFR an die HOFA mit der Anforderung «Kaltstart». Die HOFA meldet danach ihren Status «Aus», «Kaltstart init» und sendet für jeden Platz eine Belegungsmeldung. Der MFR schickt dann eine Kontrollmeldung mit der Anforderung «Warmstart», was die HOFA mit den Statusmeldungen «Aus», «Kaltstart abgeschlossen» und «Ein» quittiert. Der Ablauf wird in Abbildung 18.1 dargestellt.

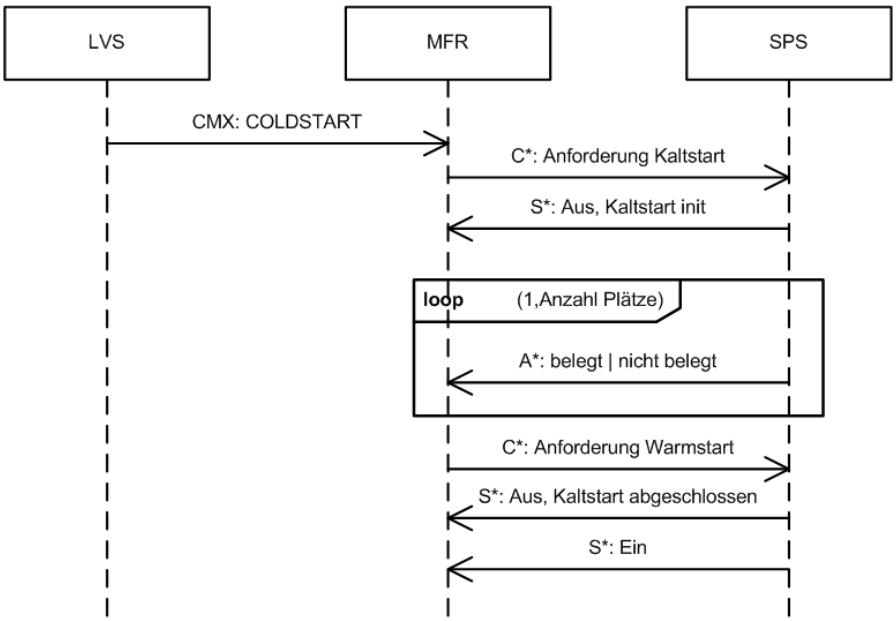


Abbildung 18.1: Kaltstart HOFA

Kiste auslagern

Das Auslagern einer Kiste erfolgt ähnlich wie das Einlagern, welches in Abschnitt 2.3 nur kommen die Telegramme für das RBG am Anfang und zum Schluss folgt ein «Kiste abgehoben». Ein Beispielablauf zeigt Abbildung 18.2.

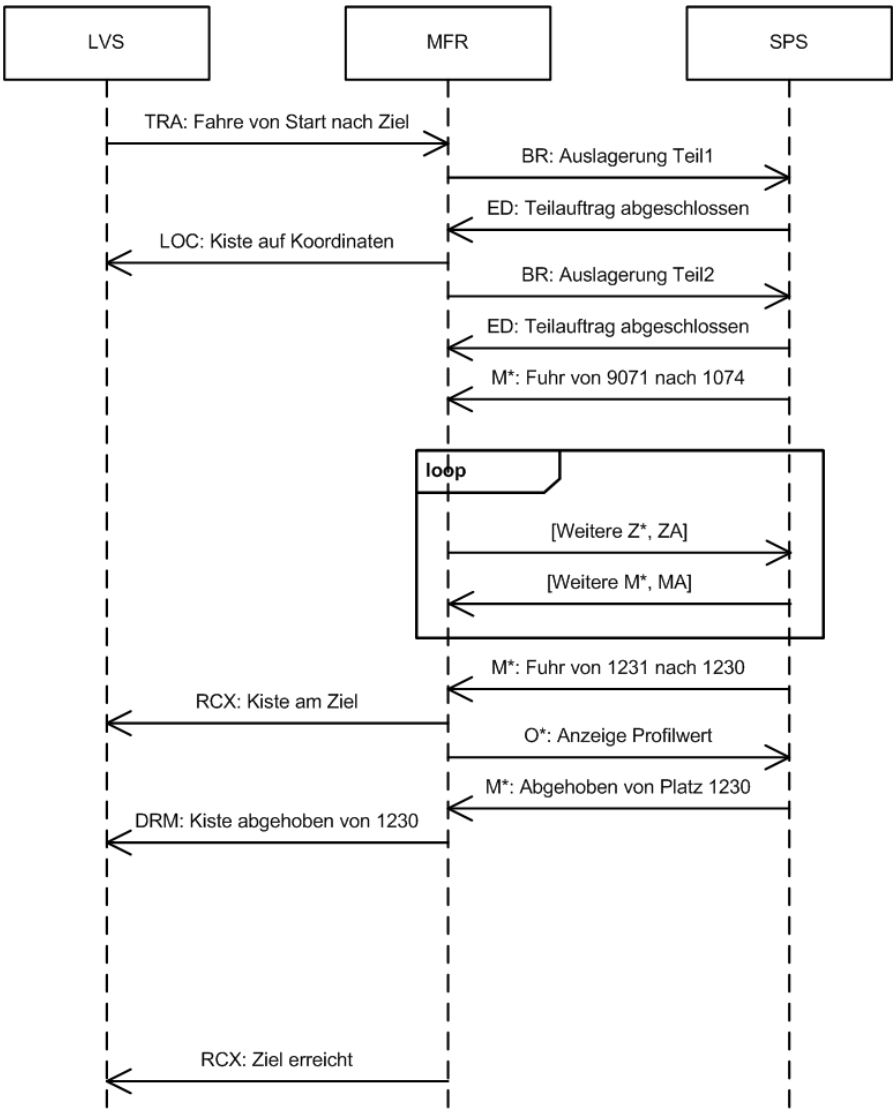


Abbildung 18.2: Kiste auslagern

Pärchenbildung

Pärchenbildung nennt man das Verfahren, dass mehrere Kisten gemeinsam transportiert werden. Dazu gibt es spezielle Plätze im Lager, an denen aufgetaktet (das Pärchen gebildet) wird und solche, an denen abgetaktet (das Pärchen aufgelöst) wird. Ein Pärchen kann dabei aus zwei oder mehreren Kisten bestehen. Auf Telegrammebene gibt es zu beachten, dass für ein Pärchen nur ein M*-Telegramm generiert wird, auch wenn mehrere Kisten den Platz wechseln. Zusätzlich können spezielle Codes verwendet werden, um Pärchen zu bilden oder aufzulösen. Nachfolgend werden die unterschiedlichen Arten zum Auf- und Abtakten beschrieben (Quelle: Mail von Tommi Lindgren).

Auftakten

Pull-Prinzip *Auf Platz A steht eine Palette p1 und auf Platz B steht eine Palette p2. A und B sind nebeneinander und B ist vor A. Wenn man will, dass p1 und p2 als Pärchen weiterfahren schickt man ein Z-Telegramm von B nach C (der Nachfolgeplatz von B) mit dem Code «bilde ein pärchen». Dann fahren beide Paletten nach C. Wenn man nur p2 bewegen will schickt man ein Z-Telegramm von B nach C mit Code «fahre einzeln». Dann fährt p2 von B nach C und p1 bleibt auf A stehen.*

Push-Prinzip *Auf Platz A steht eine Palette. Platz B ist leer. B ist der Platz auf dem augetaktet wird. Platz B hat eine maximale Kapazität von Paletten, die auf B aufgetaktet werden können. Das Prinzip ist, solange man auftakten will kann man ein Z-Telegramm von A nach B schicken. Das heisst, wenn auf Platz A eine Palette steht, wird sie auf den Platz B geschoben und bildet ein Pärchen mit allen Paletten, die auf B stehen. Wenn man will, dass die aufgetakteten Paletten weiterfahren, schickt man ein Z-Telegramm von B nach C (den Nachfolgeplatz von B).*

Beispiel: B ist leer auf A steht Palette p1.

- *Der MF schickt ein Z-Telegramm von A nach B. Nun steht Palette p1 auf B.*
- *Auf A kommt die Palette p2 an. Der MF schickt ein Z-Telegramm von A nach B. Nun stehen p1 und p2 auf B (p1 zuvordest).*
- *Auf A kommt die Palette p3 an. Der MF schickt ein Z-Telegramm von A nach B. Nun stehen p1, p2 und p3 auf B (p1 zuvordest, dann p2, dann p3).*
- *Der MF möchte weiterfahren und schickt deshalb ein Z-Telegramm von B nach C. Nun fahren p1, p2 und p3 als «Pärchen» weiter. Das Pärchen wird auf den Platz C verschoben.*

Abtakten

Automatisch

Es gibt die Abtaktplätze A, B, B' und C. Von B nach A werden die Paletten abgetaktet. B' ist ein Spezialplatz der nur dafür da ist, damit die SPS den MF mitteleilen kann, dass der Rest des Pärchens auf Platz B angekommen ist. C ist der Vorplatz von B.

1. *A und B sind am Anfang leer. Jetzt kommt ein Pärchen der Grösse n (Anzahl Paletten im Pärchen) auf Platz B an. Dies erfährt man durch ein M-Telegramm von C nach B.*
2. *Dann kommt ein M-Telegramm von B nach A. Jetzt steht die erste Palette des Pärchens auf A.*
3. *Dann kommt ein M-Telegramm von B' nach B. Jetzt steht der Rest des Pärchens (Grösse n-1) ganz vorne auf Platz B. Falls vorher nur eine Einzelpalette auf B kommt dieses Telegramm nicht. (falls n = 1)*

Die Telegramme von Schritt 2 und Schritt 3 können auch in umgekehrter Reihenfolge kommen, dies ist physikalisch bedingt.

4. *Die Pallette auf Platz A wird entfernt (entweder fährt sie weiter oder wird von der Anlage entfernt.)*

2,3 und 4 wiederholen sich n Mal, bis das ganze Pärchen abgetaktet wurde.

MF gesteuert

Die M-Telegramme kommen gleich, wie beim automatisch abtakten, nur schickt hier der MF ein Z-Telegramm von B nach A. Hier kann im Code verschlüsselt werden, wieviele Paletten abgetaktet werden.

V

Design

Verantwortlich: Rico Steffen

Datum	Version	Änderung
27. 10. 2006	0.1	Erstellt
30. 10. 2006	0.2	Architektur
30. 11. 2006	0.3	Design
02. 12. 2006	0.4	Entscheide
11. 12. 2006	0.5	Änderungen
13. 12. 2006	1.0	Final

19 Klassen

Im folgenen Abschnitt werden die Klassendiagramme aufgeführt. Die gelben Klassen sind die wichtigsten und werden kurz beschrieben. Die Roten Klassen wurden bereits im Technischen Bericht erwähnt und deshalb nochmals speziell hervorgehoben, auch sie sind kurz beschrieben. Blaue Klassen gehören teilweise zusammen und sind weniger wichtig. Die grünen Klassen sind eher unwichtig oder Hilfsklassen.

Persistency

Die Persistency ist in der Abbildung 19.1 dargestellt, wobei die Configuration in einem separaten Abschnitt erläutert wird.

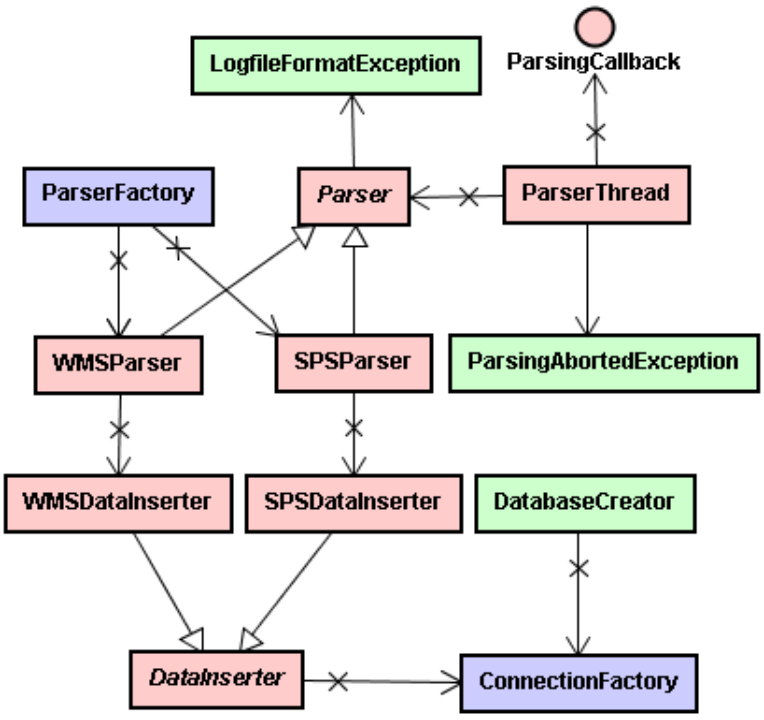


Abbildung 19.1: Persistency

Parser	Der `Parser` liest ein Logfile ein und übergibt die geparsten Telegramme dem `DataInserter`.
WMSParser	Spezifischer `Parser` für WMS-Logfiles.
SPSParser	Spezifischer `Parser` für SPS-Logfiles.
DataInserter	Der `DataInserter` wird verwendet um einzelne Telegramme, welche als `String` übergeben werden, in die Datenbank zu schreiben.
WMSDataInserter	Spezifischer `DataInserter` für WMS-Telegramme.
SPSDataInserter	Spezifischer `DataInserter` für SPS-Telegramme.
ParserThread	Der `ParserThread` wurde erstellt, um dem Benutzer das Abbrechen des Parsing-Vorganges zu ermöglichen.
ParsingCallback	Wird verwendet, um dem Aufrufer des Parsers mitzuteilen, dass der Parsing-Vorgang beendet wurde.

Configuration

Die Klassen des Configuration Paketes sind in der Abbildung 19.2 dargestellt.

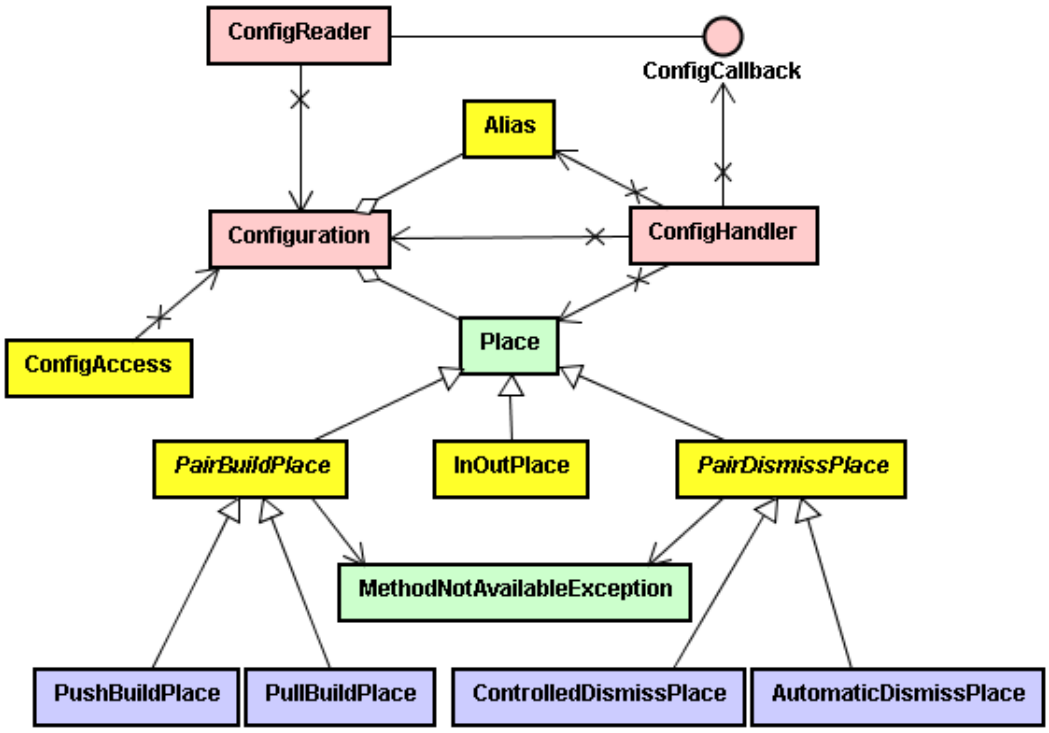


Abbildung 19.2: Configuration

ConfigReader	Der `ConfigReader` erstellt den SAX-Parser und bietet die Methode zum lesen der Konfigurationsdatei an.
ConfigHandler	Er wird vom SAX-Parser aufgerufen und erstellt die `Configuration`.
Configuration	Sie beinhaltet alle Konfigurationselemente.
ConfigCallback	Es wird verwendet, um zu signalisieren, dass die Konfiguration eingelesen wurde.
Alias	Ein `Alias` repräsentiert zwei verschiedenen Bezeichnungen für einen `Place`.
ConfigAccess	Die Klasse `ConfigAccess` bietet Methoden, um auf die Konfiguration zuzugreifen.
InOutPlace	Ein `InOutPlace` repräsentiert einen Platz im Lager, auf welchem Behälter/Paletten ein- oder ausgelagert werden.
PairBuildPlace	Ein `PairBuildPlace` ist ein Platz auf welchem Pärchen gebildet werden.
PairDismissPlace	Ein `PairDismissPlace` ist ein Platz, auf welchem Pärchen aufgelöst werden.

Problem Domain

Die Problem Domain ist in der Abbildung 19.3 dargestellt.

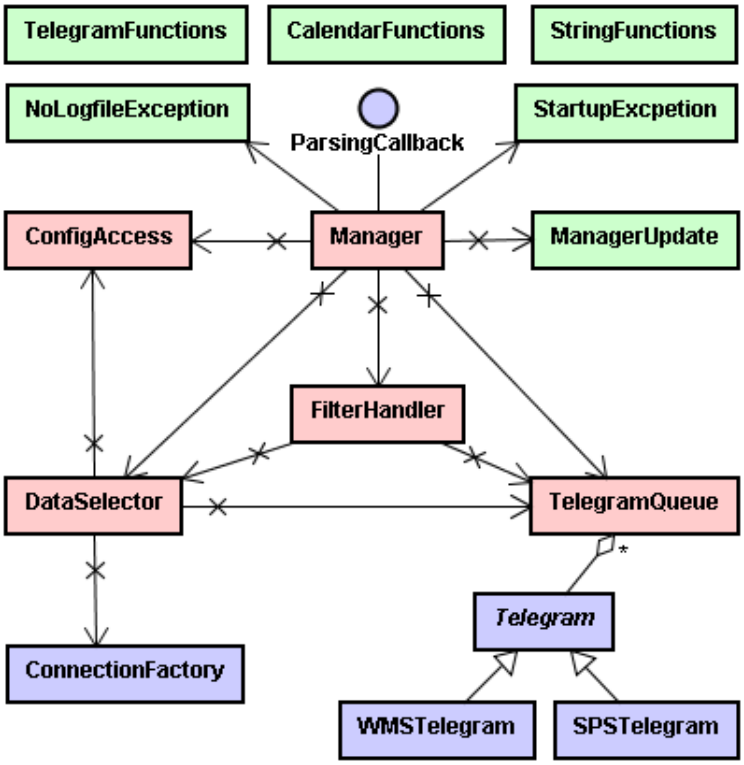


Abbildung 19.3: Problem Domain

DataSelector Der `DataSelector` wird gebraucht, um spezifische Abfragen an die Datenbank auszuführen.

ConfigAccess Diese Klasse wird gebraucht, um Konfigurationsinformationen abzufragen.

FilterHandler Der `FilterHandler` hat für jeden Filter entsprechende Methoden, in welchen er via `DataSelector` die entsprechenden Telegramme abfragt und bildet danach die Schnittemenge der gefundenen Telegramme.

Manager Der `Manager` ist die Schnittstelle zum User Interface und verwaltet die von den Filtern gefundenen Telegramme.

Telegram-Queue Die `TelegramQueue` wird verwendet, um Telegramme zu speichern.

User Interface

Die User Interface ist in der Abbildung 19.4 dargestellt.

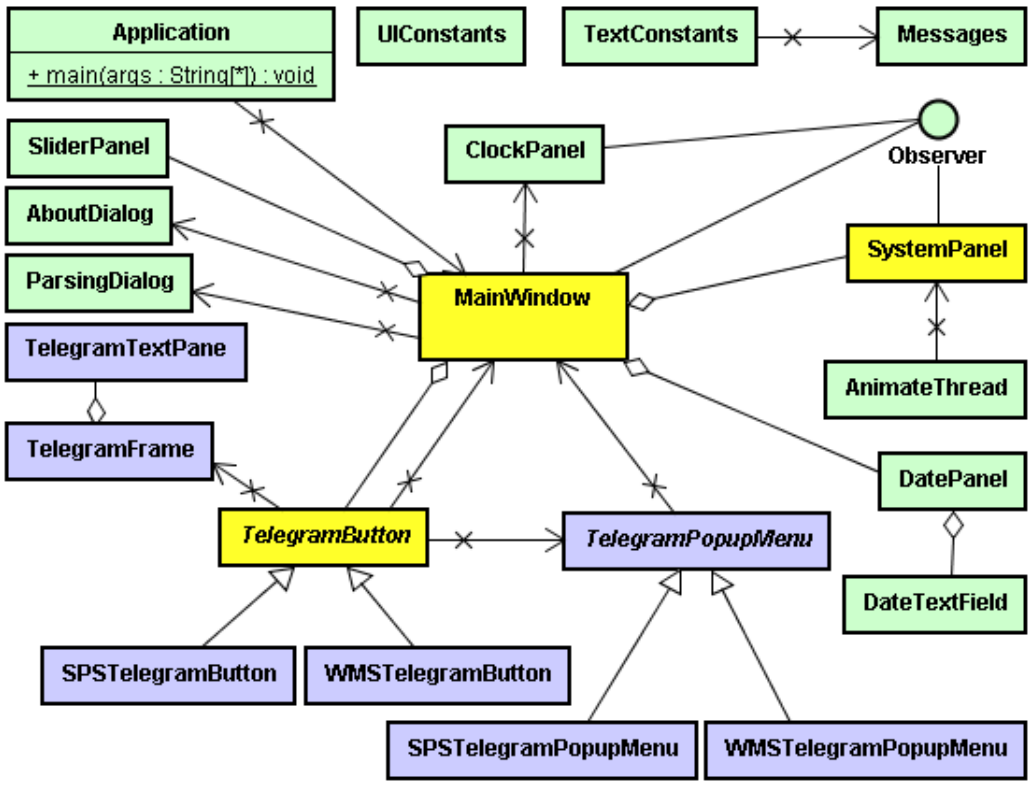


Abbildung 19.4: User Interface

MainWindow	Das `MainWindow` beihaltet alle Components des User Interfaces.
SystemPanel	Das `SystemPanel` ist für die Darstellung der gefunden Telegramme verantwortlich
TelegramButton	Ein `TelegramButton` repräsentiert ein einzelnes Telegramm im User Interface.

20 Design-Entscheide

20.1 Ablaufdesign

Im Folgenden werden zwei Design-Varianten, für den Ablauf der Logfile-Verarbeitung und der Darstellung, genauer betrachtet und dargestellt.

Einzelner Ablauf

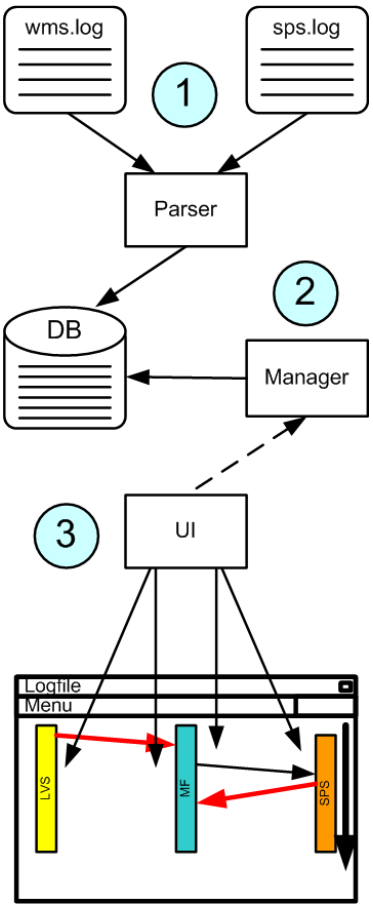


Abbildung 20.1: Einzelner Ablauf

Abbildung 20.1 zeigt den einzelnen Ablauf. Nachfolgend werden die Punkte erläutert:

1. Die beiden Logfiles werden von einem Parser eingelesen und die Telegramme in eine Datenbank geschrieben.
2. Der Manager führt Abfragen auf die Datenbank aus und enthält die Telegramme dieser Abfrage in einer einzelnen sortierten Liste.
3. Das User Interface gib dem Manager den Auftrag für eine Abfrage, erhält das Ergebnis und stellt die Telegramme auf der gewünschten Seite dar.

Paralleler Ablauf

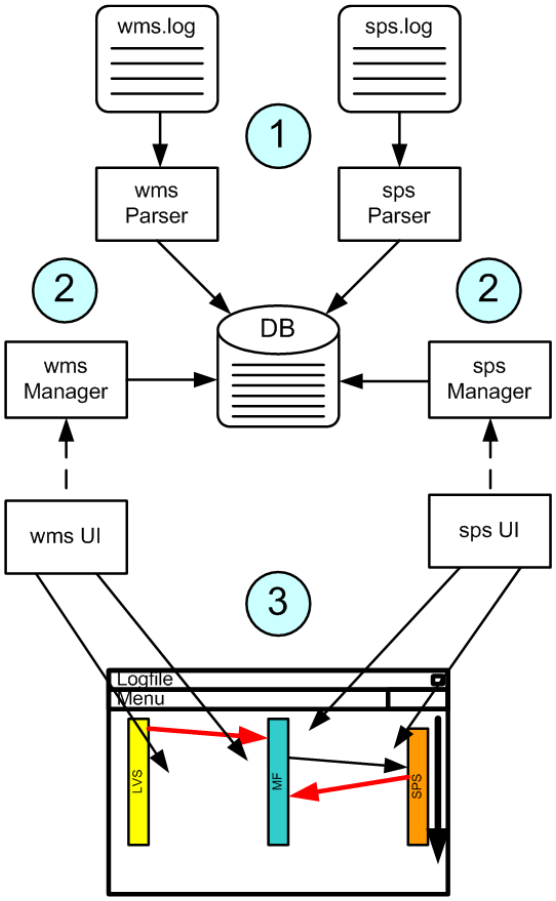


Abbildung 20.2: Paralleler Ablauf

In der Abbildung 20.2 wird der parallele Ablauf gezeigt. Die einzelnen Punkte:

1. Die beiden Logfiles werden von je einem separaten Parser eingelesen und die Telegramme in eine Datenbank geschrieben.

2. Pro Telegramm-Typ existiert ein Manager, welcher Abfragen auf die Datenbank ausführt.
3. Das User Interface ist ebenfalls auf die Telegramm-Typen aufgeteilt und stellt jeweils nur die zuständigen Telegramme dar.

Entscheidung

In der ersten Implemantations-Phase legten wir uns noch auf keinen der beiden Abläufe fest. Wir implementierten zuerst nur die SPS-Seite und hofften, dadurch bessere Erkenntnise zu erlagen, welche der beiden Lösungen die bessere ist.

In einer weiteren Phase wurde ein Mittelweg gewählt. So, dass der Manager zwei Telegrammlisten parallel führt. Da diese Lösung aber keine Vorteile brachte wurde im finalen Design der «Einzelne Ablauf» implementiert. Für das Einlesen der Logfiles werden aber trotzdem zwei separate Parser benutzt.

20.2 Verwaltung der Logeinträge

Die Überlegungen und Entscheidungen bezüglich der Verwaltung der Logeinträge werden im folgenden Abschnitt genauer erläutert.

Javaobjekte

In der ersten Implementations-Phase wurden die vom Parser eingelesen Telegramme als Objekte in einer Queue gespeichert. Diese Liste erreicht eine enorme Grösse, falls Logfiles mit mehreren tausend Einträgen eingelesen werden. Somit steigt auch der Speicherverbrauch der ganzen Applikation. Um die Entwicklung eines Paging-Mechanismus zu umgehen und zur einfacheren Implementierbarkeit der Filter stiegen wir auf die Lösung mit einer Datenbank um.

Datenbank

Die geparsten Logfile-Einträge werden in eine Datebank geschrieben. Dabei werden noch keine Objekte für die Telegramme erstellt. Die Datenbank erlaubt es dann realtiv einfach, nur die benötigten Telegramme aus den Tabellen per `SELECT`-

Abfragen herauszulesen. Somit werden nur für die gefunden Telegramme Objekte erstellt und nicht für alle, welche im Logfile stehen.

SQLite Als Datenbank verwerdenten wir die OpenSource Datenbank SQLite[1]. Die Wahl fiel aus folgenden Gründen auf SQLite [Owe06]:

- Keine Installation notwendig, es wird nur eine DLL benötigt.
- Der Zugriff geht über JDBC, so kann ohne Probleme auch eine andere Datenbank verwendet werden, für welche JDBC-Treiber vorliegen.
- Die Datenbank benötigt wenig Speicherplatz und verfügt über eine gute Performance.

Auf die Verwendung von Oracle[2] wurde verzichtet. Somit war keine Datenbank-Installation auf unseren Systemen nötig, aber, falls gewünscht, kann über JDBC auch eine Oracle-DB angesprochen werden.

20.3 User Interface

Das im Kapitel 3.3 skizzierte User Interface (Abbildung 3.1) wurde als Grundlage für das implementierte Design verwendet und entsprechend dem im Kapitel 14 beschriebenen Use Case erweitert.

20.4 Telegrammdarstellung

Ablauf

Damit der Zeitliche Ablauf der Telegramme ersichtlich ist, wurden die im ersten Desing die Abständen zwischen den Telegrammen im Verhältnis zur verstrichenen Zeit dargestellt. Diese Art der Darstellung stellte sich jedoch schnell als unübersichtlich und platzraubend dar. So werden die Telegramm im Schluss-Design nur noch chronologisch – in der richtigen Reihnenfolge – dargestellt.

[1] SQLite 3.3.8: www.sqlite.org
[2] Oracle: www.oracle.com

Implementation

Für die einzelnen, darzustellenden Telegramme wurde ein eigener `JButton` implementiert. Dadurch sind die Mausaktionen eifach zu verwalten. Da bei der Animation der Telegramme Java Swing sehr langsam neu zeichnet, wird für die Zeichnung der Animation und der Systeme (LVS, MFR und SPS) Java2D[3] verwendet. Sobald mehr als 200 Telegramme visualisiert werden sollen, wird die Animation abgeschalten, damit das Neuzeichnen des User Interfaces nicht zu lange dauert.

[3]Java2D: http://java.sun.com/products/java-media/2D/

VI

Test

Verantwortlich: Roland Fehlmann

Datum	Version	Änderung
06. 12. 2006	0.1	Erstellt
10. 12. 2006	0.2	Vorgehen
12. 12. 2006	0.3	Ebenen
13. 12. 2006	0.3	Überarbeitung
13. 12. 2006	1.0	Final

21 Eingesetzte Hilfsmittel

21.1 Pair Programming

Die meisten Programmier-Aufgaben werden unter den Teammitgliedern aufgeteilt, damit ein gutes Vorankommen garantiert ist. Bei besonders komplizierten oder fehleranfälligen Methoden und Programmteilen wird gemeinsam gearbeitet. So können Fehler vermieden werden und die Qualität des Codes wird verbessert.

Bei unvorhergesehenen Schwirigkeiten, zum Beispiel bei heiklen Codestellen, wird das Teammitglied kurz hinzugezogen und man schaut sich den Code zusammen an.

21.2 Coderichtlinien

Am Projektanfang wurde ein Code Template für Eclipse erstellt. Damit wird sichergestellt, dass beide Teammitglieder sauber formatierten Code schreiben. Zudem kann der Code durch Eclipse automatisch formatiert werden.

21.3 Eclipse

Das Eclipse ist eine gute Hilfe um besseren Code zu schreiben. Es bietet die Möglichkeit, Warnhinweise für unschönen Code, nicht verwendete Felder und Methoden oder Anderes, einzuschalten. Dies ermöglicht es, relativ einfach den Code zu überprüfen und auch undokumentierte Stellen zu finden.

21.4 Metrics

Metrics[1] ist ein Eclipse Plugin, welches einem verschiedene Codeanalysen zur Verfügung stellt. So zum Beispiel die Anzahl Codezeilen, Komplexitäts-Analysen oder die Anzahl Parameter einer Methode. Dadurch bietet es eine einfache Möglichkeit, komplexe Methoden aufzuspüren, welche nochmals genauer betrachtet und gegebenenfalls einem Refactoring unterzogen werden müssen.

[1] Metrics: metrics.sourceforge.net

22 Durchzuführende Tests

22.1 Unit Tests

Mit Unit Tests soll die Kernfunktionalität des persistency- und des problem domain-Packages getestet werden. Die Tests müssen sicherstellen, dass Änderungen am Code die Funktionalität der Klassen nicht beeinträchtigen.

22.2 Ad-hoc Tests

Ad-hoc Test sind informelle Tests, welche während der Implementation durch den Entwickler durchzuführen sind. Solche Tests werden während der Programmierung ständig durchgeführt, um erstellte Funktionen zu überprüfen.

22.3 Codeanalysen

Der Code sollte mit Eclipse, Metrics und Code-Reviews überprüft werden. Warneinstellungen im Eclipse: Alle vorhandenen Checks werden aktiviert. Einzige Ausnahmen:

- Unqualified access to instance field
- Parameter assignment

Im Metrics soll überprüft werden:

- Number of Parameters max. 5
- McCabe Cyclomatic Complexity max. 10
- Nested Block Depth max. 5

22.4 Usability Tests

Durch das Durchführen von Usability Tests soll sichergestellt werden, dass der spätere Benutzer mit der Bedienung der Software zurecht kommt und die Bedienung der Software intuitiv ist. Es kann davon ausgegangen werden, dass der spätere Anwender selber Software entwickelt und über entsprechende Kentnisse in der Bedienung eines Programmes verfügt.

23 Durchgeführte Tests und Resultate

23.1 Unit Tests

Für folgende Klassen wurden Unit-Tests geschrieben:

- `ConfigReader`
- `DataInserter`
- `DataSelector`

- `Parser`
- `SPSParser`
- `WMSParser`

- `Manager`
- `FilterHandler`

Die Testklassen befinden sich jeweils in einem eigenen Test-Package:

- `ch.hsr.lfv.pe.tests`
- `ch.hsr.lfv.pd.tests`

Bezüglich Testabdeckung wurde eine Analyse mit dem Eclipse Plugin Coverlipse[1] durchgeführt. Tabellen 23.1 und 23.2 zeigen die Resultate.

23.2 Ad-hoc Tests

Ad-hoc Tests wurden nach Gutdünken der Entwickler durchgeführt. Es wurden keine Protokolle darüber geführt.

[1] Coverlipse: coverlipse.sourceforge.net

Package	Abdeckung	Kommentar
`auxiliaryFunctions`	67%	`TelegramFunctions` wird nur zu einem Drittel durchlaufen.
`playback`	97%	Im `FilterHandler` werden nicht ganz alle Äste durchlaufen.
`telegrams`	82%	Ein paar get()-Methoden sowie die equals()-und compareTo()-Methoden von `Telegram` werden nicht aufgerufen.

Tabelle 23.1: Report für Package `pd`

Package	Abdeckung	Kommentar
`config`	82%	`ConfigAccess` wird nur zu zwei Dritteln durchlaufen, im ConfigReader werden nicht alle catch()-Blöcke durchlaufen.
`config.elements`	80%	Nur simple get()- und set()-Methoden werden nicht durchlaufen.
`database`	90%	Hauptsächlich im `DataSelector` wird nicht der ganze Code durchlaufen.
`parser`	82%	Mehrheitlich catch()-Blöcke werden nicht durchlaufen.

Tabelle 23.2: Report für Package `pe`

23.3 Codeanalysen

Die Bedingungen, welche mit Eclipse überprüft wurden, werden alle erfüllt. Metrics meldete Probleme für die unten aufgeführten Methoden.

Number of Parameters

- `DataSelector.getSql` (Parameter: 6)

Die Methode `getSql` ist eine private Hilfsmethode und wird nur von den spezifischeren Methoden `getSPSSql` und `getWMSSql` verwendet.

McCabe Cyclomatic Complexity

- `UIConstants.getTelegramColor` (Komplexität: 35)
- `MainWindow.setFilters` (16)
- `MainWindow.update` (11)

- `ConfigHandler.characters` (16)
- `ConfigHandler.startElement` (15)
- `ConfigHandler.endElement` (12)

- `SPSLogGenerator.getRLType` (17)
- `SPSLogGenerator.getLRType` (14)

Alle oben erwähnten Probleme sind auf grosse switch-case- oder if-else-Statements zurückzuführen, die jedoch nicht tief verschachtelt sind. Dadurch stellen sie für die Verständlichkeit des Codes kein Problem dar.

Nested Block Depth

Es gibt keine Methode, welche diese Bedingung verletzt.

23.4 Usability Tests

Die Bedienbarkeit der Applikation wurde bei der Entwicklung früh überprüft und optimiert. Beim Testen der Funktionalität achteten wir auch gleich darauf,

dass das User Interface intuitiv bedient werden konnte. Da wir die Software selber einsetzten, um Logfiles anzuschauen, merkten wir relativ gut, was wir noch zu verbessern hatten.

Zusätzlich wurde während den Demonstrationen bei der Firma Stöcklin Software darauf geachtet, wie die Mitarbeiter die Appklikation bedienen wollten. Wenn sie eine Funktion falsch anwendeten oder uns fragen mussten, wie ein bestimmtes Verhalten des Programms hervorgerufen werden konnte, nahmen wir uns diese Informationen als Input und versuchten, das Programm entsprechend anzupassen.

Es wurden jedoch keine im Voraus definierten Usability-Tests mit den Mitarbeitern von Stöcklin Software durchgeführt.

VII

Persönliche Erfahrungen

Verantwortlich: Roland Fehlmann & Rico Steffen

Datum	Version	Änderung
06. 12. 2006	0.1	Erstellt
11. 12. 2006	0.2	Rico
12. 12. 2006	0.3	Roland
13. 12. 2006	1.0	Final

24 Roland Fehlmann

Die letzten acht Wochen waren interessant, lehrreich und vergingen wie im Flug; kaum dass wir mit der Diplomarbeit begonnen haben, ist sie auch schon wieder fertig.

Da die Diplomarbeit meine letzte Tat ist in diesem Studium, startete ich mit einer grossen Portion Motivation. Doch nicht nur die Tatsache, das Studium bald zu beenden, sondern auch die Zusammenarbeit mit Stöcklin Software und die damit verbundene Gewissheit, etwas Produktives zu leisten, hatten ihren Anteil daran. Hinzu kam, dass ich wieder einmal mit Rico zusammenarbeiten konnte.

Die Arbeit selbst verlief relativ unspektakulär. Die in den Studien- und anderen Projektarbeiten gewonnene Routine trug dazu bei, dass wir gut abschätzen konnten, was uns erwartete. Einzige Unbekannte stellte die Zusammenarbeit mit einem Industriepartner dar, dank der kooperativen Art der Stöcklin Software-Mitarbeiter erwuchsen daraus aber keine Probleme.

Die Zusammenarbeit klappte sowieso mit allen Beteiligen optimal, neben den Leuten von Stöcklin verlief auch der Umgang mit Rico, wie so oft, reibungslos. Ebenso das Verhältnis zu unserem Betreuer Daniel Keller – ich hatte das Gefühl viele Freiheiten und auch Vertrauen zu besitzen. Es war angenehm, unter diesen Verhältnissen zu arbeiten.

Bleibt die Frage nach der Zufriedenheit am Ende der Diplomarbeit: Vorhanden. Nicht nur wegen der gelungenen Software (obwohl nicht alles perfekt ist), sondern auch wegen der gewonnenen Erfahrung und der Erkenntnis, während längerer Zeit nicht viel anderes zu tun als Software zu entwickeln, ohne die Freude daran zu verlieren. Sicherlich kein schlechtes Omen für einen zukünftigen Job in dieser Branche.

25 Rico Steffen

Ich war motiviert in die letzten acht Wochen meines Studiums gestartet. Freute mich auf die Diplomarbeit und auch jetzt am Ende ist meine Stimmung, in Bezug auf die Arbeit, noch immer sehr gut.

Anfangs hiess es für uns Einarbeiten in das Thema der Lagerverwaltung. Die Mitarbeiter der Stöcklin Software waren uns dabei eine grosse Hilfe und gaben uns besonders am Anfang eine gute Einführung. Dennoch waren die vielen verschiedenen Telegramme für mich zuerst noch ein chaotischer Haufen. Mit der Simulations-Software, einer genauen Analyse der Telegramme und unserer eigenen Visualisierung löste sich das Chaos jedoch bald auf.

Mit der Besichtigung des Manor-Lagers in Hochdorf bekamen wir dann noch einen realen Eindruck eines solchen Lagerverwaltungssystems. Leider waren viele Teile der Anlage noch nicht in Betrieb.

Die Zusammenarbeit mit Stöcklin und dem Betreuer lief sehr gut. Dies lag wohl vor allem daran, dass bei der Stöcklin viele ehemalige Studenten arbeiten und mit der Arbeit vertraut sind. Während den Besprechungen mit dem Betreuer wurden die wichtigsten Dinge angeschaut, aber ansonsten wurde uns freie Hand gewährt, was ich sehr positiv fand.

Zum Schluss bin ich mit unserer Arbeit sehr zufrieden. Die Zusammenarbeit zwischen Roland und mir klappte wieder hervorragend, wie schon in der letzten Studienarbeit und wir kamen, trotz der kurzen Zeit von acht Wochen, nie in einen grossen Arbeitsstress.

VIII

Benutzerhandbuch

Verantwortlich: Rico Steffen

Datum	Version	Änderung
06. 12. 2006	0.1	Erstellt
11. 12. 2006	0.2	Installation
11. 12. 2006	0.3	Bedienung
12. 12. 2006	0.4	Entwickler
13. 12. 2006	1.0	Final

26 Installationsanleitung

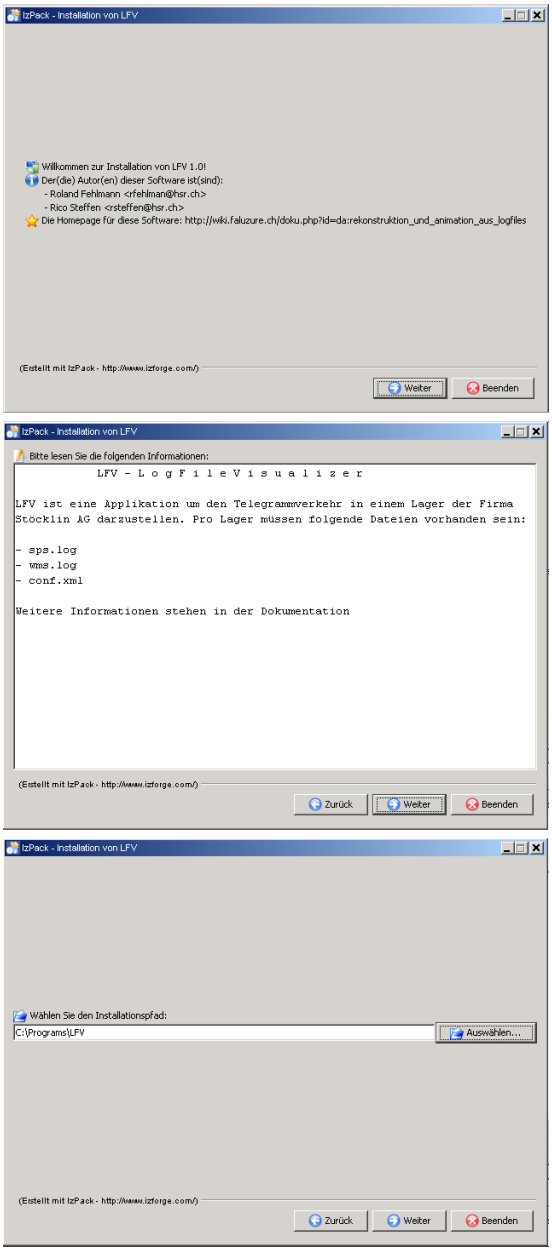


Abbildung 26.1: Installation Schritte 1-3

Die Installation gestaltet sich sehr einfach. Nachdem man das Installations-JAR-File ausgeführt hat – je nach Installation der Java-Umgebung[1] per Doppelklick oder aus der Kommandozeile mit

```
java -jar LFVInstall.jar
```

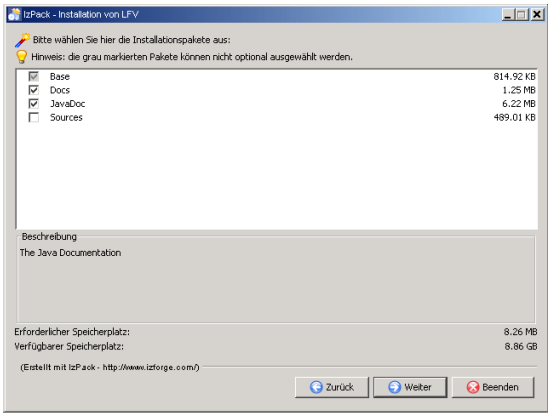
erscheint der IzPack-Installer. Klicken auf «Weiter» bewirkt die Fortsetzung der Installation. Ein Klick auf «Beenden» beendet die Installation und belässt das System unverändert.

Beim nächsten Fenster lesen Sie die Readme-Datei. Danach können Sie mit «Weiter» die Installation fortsetzen. Auch hier haben Sie die Möglichkeit, die Installation abzubrechen. Mit einem Klick auf «Zurück» kommen Sie zurück zum Willkommens-Bildschirm.

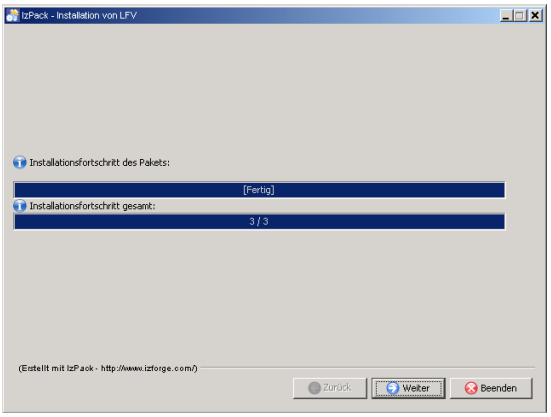
Im dritten Installationsfenster kann spezifiziert weden wohin, LogFileVisualizer installiert werden soll. Als Standardpfad wird das Standardverzeichnis für Programme vorgeschlagen. Sie können natürlich einen beliebigen anderen Pfad angeben.

[1] Java Runtime 5 muss installiert sein. Download: `http://java.com/en/download/windows_xpi.jsp`

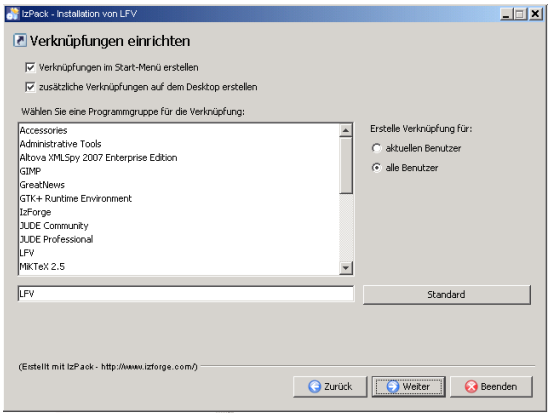
Das vierte Fenster erlaubt es einem, die gewünschten Komponenten zu Installieren. Base ist nötig, die anderen 3 Punkte sind freiwillig. Für Anwender wird Base und Docs empfohlen, Entwickler müssen mindestens die Sources dazuinstallieren. JavaDoc wird für Enwickler zusätzlich empfohlen.



Als Nächstes folgt das Kopieren der Dateien. Bitte warten Sie, bis alle Komponenten kopiert wurden (Statusbalken beachten). Danach können Sie mit «Weiter» mit der Installation fortfahren.



Als letzter wirklicher Schritt folgt das Erstellen von Verknüpfungen. Bitte geben Sie an, wo und welche Verknüpfungen Sie haben möchten – wenn Sie nicht ganz sicher sind, dürften die Voreinstellungen keine schlechte Wahl sein.



Danach folgt nur noch der Bildschirm mit der Bestätigung, dass die Installation erfolgreich war. Klicken Sie auf «Fertig» um den Installer zu schliessen.

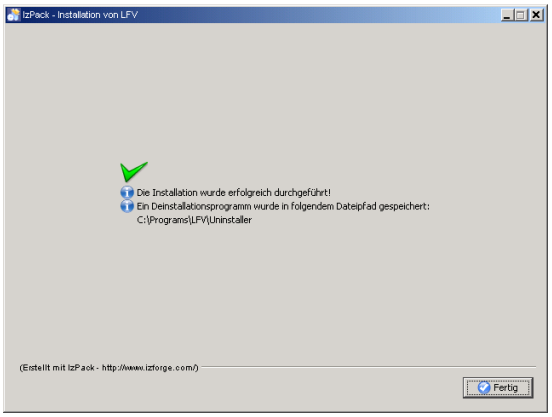


Abbildung 26.2: Installation Schritte 4-7

27 Bedienungsanleitung

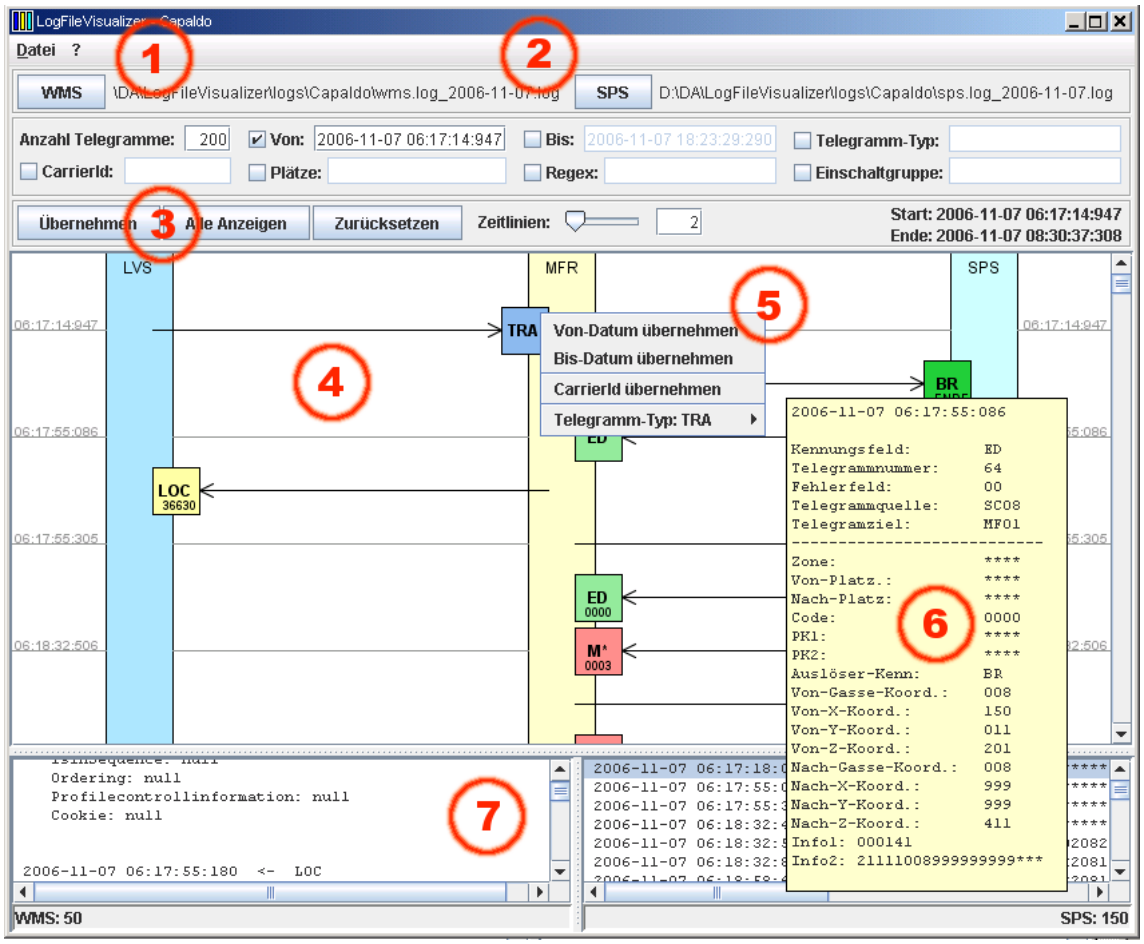


Abbildung 27.1: Screenshot der Applikation

Abbildung 27.1 zeigt die Benutzeroberfläche von LogFileVisualizer. In den folgenden Abschnitten werden die einzelnen Komponenten (im Bild eingezeichnet: Punkte 1-7) detailliert beschrieben.

Punkt 1: Menu

Menu «Datei» In diesem Menu befindet sich je ein Eintrag um eine neues WMS- bzw. SPS-Logfile einzulesen. Ein weitere Eintrag, «neue Config-

Datei», ermöglicht das Einlesen einer neuen Konfiguration. Der letzte Eintrag im Menu, «Exit», beendet die Applikation.

Punkt 2: Logfile-Auswahl

WMS- und SPS-Button Diese zwei Buttons dienen dazu, ein neues WMS- bzw. SPS-Logfile einzulesen. Danach wird der Pfad der Datei angezeigt.

Punkt 3: Filter

Alle Filter, ausser «Anzahl Telegramme», können aktiviert bzw. deaktiviert werden. Werden mehrere Filter aktiviert, wird die Schnittmenge der gefunden Telegramme angezeigt.

Anzahl Telegramme Mit diesem Filter wird die maximale Anzahl der angezeigten Telegramme festgelegt.

Von, Bis Mit dem Von- bzw. Bis-Datum wird das früheste bzw. späteste Datum definiert.

Telegramm-Typ Mit diesem Filter kann nach Telegramm-Typen gesucht werden. Es können auch mehere Typen angegeben werden, diese müssen durch Kommas getrennt werden (z.B. TRA,M*).

CarrierId Wird eine CarrierId angegeben, werden alle Telegramme angezeigt, welche mit dieser Id in Verbindung stehen.

Plätze Hiermit kann nach Telegrammen zu bestimmten Plätzen gefiltert werden. Es können ebenfalls mehrere Plätze, durch Kommas getrennt, angegeben werden (z.B. 1208,1209).

Regex Der Regex-Filter bietet die Möglichkeit, über die Logfile-Zeile (siehe Punkt 7: *Telegramm-Zeile* unten) eine Suche mit Regulären Ausdrücken durchzuführen.

Einschaltgruppe Mit diesem Filter werden alle Telegramme zu einer bestimmten Einschaltgruppe angezeigt. Dies ist vor allem zur Beobachtung eines Kaltstarts hilfreich.

Übernehmen Mit diesem Button werden die Filtereinstellungen übernommen und die gefundenen Telegramme angezeigt. Dies kann auch in jedem Filterfeld durch betätigen der «Enter»-Taste ausgelöst werden.

Alle Anzeigen Damit werden alle Telegramme, die in den Logfiles vorhanden sind, angezeigt. Aus Performance-Gründen werden jedoch nur maximal 2000 Telegramme gezeichnet.

Zurücksetzen Die Telegramm-Darstellung (Punkt 4: *Telegramm-Darstellung*, unten) wird gelöscht und keine Telegramme werden mehr angezeigt.

Zeitlinie Dieser Slider bestimmt, bei jedem wievielten Telegramm eine Zeitlinie in der Telegramm-Darstellung (Punkt 4: *Telegramm-Darstellung*) gezeichnet werden soll.

Start, Ende Hier wird die Zeit des ersten bzw. letzten angezeigten Telegramms aufgeführt.

Punkt 4: Telegramm-Darstellung

Telegramme In diesem Bereich werden die Telegramme grafisch dargestellt. Die Senderichtung ist mit einem Pfeil angegeben und der Empfänger ist jeweils das System auf welchem das Telegramm liegt.

Systeme Die Systeme (LVS, MFR und SPS) werden im Hintergrund dargstellt. Damit ist ersichtlich, wohin welches Telegramm gesendet wurde.

Punkt 5: Popupmenu

Wird auf ein Telegramm mit der rechten Maustaste geklickt, so erscheint ein Popup-Menu. Dieses Menu variiert je nach Telegramm-Typ. Die so gesetzen Filter werden automatisch aktiviert.

Von-, Bis-Datum Damit kann das Von- bzw. Bis-Datum in den entsprechen Filter übertragen werden.

CarrierId Ist in einem Telegramm eine CarrierId vorhanden, kann diese ebenfalls in den Filter übernommen werden.

von- und nach-Platz Der von- bzw. nach-Platz kann bei SPS-Telegrammen übernommen oder zur Liste hinzugefügt werden.

Telegramm-Typ Jeder Telegramm-Typ kann in den Filter übernommen oder angefügt werden.

Punkt 6: Telegramm

Wird auf ein Telegramm mit der linken Maustaste geklickt, erscheint dieses Fenster. Es verschwindet, sobald ein Mausklick darauf ausgeführt wird oder wenn die Maus das Fenster verlässt.

Zeit	Als oberste Zeile wird die genaue Zeit, zu der das Telegramm gesendet wurde, angezeigt.
Inhalt	Die restlichen Zeilen zeigen den gesamten Inhalt des Telegramms an.

Punkt 7: Telegramm-Zeile

In diesen Feldern werden die oben dargestellten Telegramme in Textform aufgeführt. Wird ein Telegramm in der *Telegramm-Darstellung* ausgewählt, wird es im entsprechend Feld markiert.

WMS-Seite	Die WMS-Telegramme werden, wie im Logfile selbst, mehrzeilig dargestellt.
SPS-Seite	Ein SPS-Telegramm wird genau auf einer Zeile dargestellt.
Anzahl Telegramme	Hier wird jeweils die Anzahl der gefundenen Telegramme pro Seite aufgeführt.

28 Entwickleranleitung

In diesem Kapitel wird beschrieben, welche Dateien, Klassen und Methoden angepasst werden müssen, wenn die Applikation an eine neue Anlage angepasst werden muss, neue Filter implementiert werden sollen oder wenn das Format der Logfiles ändert.

28.1 Neue Anlage

Anpassungen, welche gemacht werden müssen, falls die Applikation für eine neue Anlage konfiguriert werden soll, werden in diesem Abschnitt beschrieben.

Konfigurationsfile erstellen

Für die Anlage muss ein neues Konfigurationsfile erstellt werden. In der Konfiguration werden der Anlagenname, die Pärchen-Plätze, die Ein- und Auslagerungsplätze sowie die Aliase definiert.

Code-Anpassungen

Falls die Abläufe nicht richtig dargestellt werden, sind Anpassungen am Source-Code nötig. Nachfolgend sind die wichtigsten Klassen aufgeführt, wo Anpassungen nötig sein könnten oder die für das Verständnis wichtig sind.

Parser Falls das Format der Logfiles nicht gleich ist, bzw. falls irgendwelche spezielle Einträge in den Logfiles nicht geparst werden können, muss der entsprechende Parser angepasst werden. Die Klassen sind `WMSParser` und `SPSParser`. Anpassungen müssen ev. an der Methode `checkLogFormat()` gemacht werden (falls eine Fehlermeldung “Falsches Logfile Format” kommt) oder aber ausgehend von der Methode `readFile()` durchgeführt werden.

Manager Die Klasse `Manager` ist der Anlaufpunkt für alle Methoden, welche vom UI aus aufgerufen werden. Hier sollten aber keine Änderungen nötig sein.

FilterHandler Im `FilterHandler` ist der Ausgangspunkt für alle Filter. Vom `Manager` wird die Methode `applyFilters()` aufgerufen, welche dann ihrerseits die Filter aufruft. In den einzelnen Filtermethoden könnten einzelne Anpassungen nötig sein.

DataSelector Zusammen mit dem `FilterHandler` wird der `DataSelector` die Klasse sein, welche am ehesten angepasst oder erweitert werden muss. Der `DataSelector` ist für alle Datenbank-Abfragen zuständig und dürfte der Punkt sein, wo die falschen Telegramme geliefert werden. Je nach Filter werden andere Methoden aufgerufen, wichtig für das Verständnis sind `getWMSSql()` und `getSPSSql()`, welche jeweils die `SELECT`-Statements ausführen und die Telegramme zurückliefern.

Telegram-Queue Die Klasse `TelegramQueue` ist eigentlich ein `TreeSet`, welches Telegramme aufnehmen kann. Die `TelegramQueue` wird verwendet, um gefundene Telegramme zu übergeben.

28.2 Neuer Filter

Anpassungen, welche gemacht werden müssen, wenn ein neuer Filter implementiert werden soll.

Neue Konfigurationsfile-Einträge

Falls der neue Filter Elemente verwendet, die konfiguriert werden müssen, aber nicht in der Konfiguration vorgesehen sind, muss das XML-Schema `config.xsd` angepasst werden und die bestehenden Konfigurationsfiles müssen möglicherweise erweitert werden. Weiter müssen Änderungen an folgenden Klassen durchgeführt werden:

Configuration Die Klasse `Configuration` muss erweitert werden, damit sie die neuen Elemente aufnehmen kann.

ConfigAccess In der Klasse `ConfigAccess` müssen die Funktionen hinzugefügt werden, welche zum Zugriff auf die neuen Elemente nötig sind.

ConfigHandler Der `ConfigHandler` muss angepasst werden, damit die neuen Elemente eingelesen werden.

Code-Erweiterungen

Folgende Klassen müssen erweitert werden, um einen neuen Filter verwenden zu können:

MainWindow In der Klasse `MainWindow` muss die Methode `initFilterPanel()` erweitert werden, damit UI-Elemente zur Aktivierung und Konfiguration hinzugefügt werden. Natürlich kann der Filter auch auf andere Art in das GUI integriert werden. In der Methode `setFilters()` wird geschaut, ob der Filter aktiv ist und falls Filterwerte gesetzt sind, werden sie im `Manager` gesetzt.

Manager Der `Manager` muss um Methoden erweitert werden, welche das setzen von Filterwerten erlauben.

FilterHandler Im `FilterHandler` ist eine neue Filter-Methode zu erstellen. Damit sie aufgerufen wird, muss die Methode `getFilterQueues()` erweitert werden.

DataSelector In der Klasse `DataSelector` müssen die neuen Methoden erstellt werden, welche die entsprechenden Telegramme aus der Datenbank lesen.

28.3 Änderung im Logfile-Format

Bei einer Änderung des Logfile-Formates kommt es darauf an, wie gross die Änderung ist. Falls der Inhalt der protokollierten Daten so stark ändert, dass das Format der Datenbank angepasst werden muss, könnten grosse Änderungen an vielen Teilen der Applikation nötig sein (falls z.B. benötigte Informationen nicht mehr vorhanden sind). Dieser Fall wird hier nicht näher beschrieben, da man wohl mit der gesamten Applikation vertraut sein müsste, um sie in diesem Fall anpassen zu können. Falls die Datenbank gleich bleibt, sind folgende Klassen anzupassen:

Parser Die Klassen `WMSParser` und `SPSParser` müssen angepasst werden (je nachdem, welches Logfile-Format geändert hat). Betroffen ist die Methode `checkLogFormat()` und ausgehend von der Methode `readFile()` alle Methoden, welche von der Änderung betroffen sind.

Falls die Datenbank erweitert wird, die bestehenden Felder aber nicht ändern, müssen zusätzlich folgende Klassen angepasst werden:

DatabaseCreator Der `DatabaseCreator` muss angepasst werden (die beiden Methoden, in welchen die Tabellen erstellt werden: `makeSPSTable()` und `makeWMSTable()`).

DataInserter Die Klassen `WMSDataInserter` und `SPSDataInserter` müssen angepasst werden. Die betroffene Methode: `insertTelegram()`.

IX

Anhang

A Verwendete Software

Name:	**Version:**
Altova XMLSpy	2007
Coverlipse (Eclipse Plugin)	0.9.5.3
DokuWiki	2006-11-06
Eclipse	3.2.1
IzPack Installer	3.9.0
Java	1.5.0-09
JUDE Community/Professional	3.1
Metrics (Eclipse Plugin)	1.3.6
Microsoft Visio	2003 SP2
Microsoft Windows XP Professional	SP2
MiKTeX LaTeX	2.5
OpenOffice.org	2.0.4
SQLite	3.3.8
SubVersion	1.3.2
TeXnicCenter	1 Beta 7.01
Tortoise Svn	1.4.0

Tabelle A.1: Verwendete Software

B Danksagungen

Wir danken folgenden Personen:

- **Daniel Keller** für die Zeit, die er für unsere Betreuung aufgewendet hat.
- **Roman, Tommi und Tom**, welche sich immer Zeit nahmen, wenn wir mit Fragen zu ihnen kamen.
- **Björn** der uns bei allen LaTeX Fragen half.
- **Georg** mit seinem **Reporter-Tool** → reporter.rwf.ch, da es uns eine einfache Zeiterfassung ermöglichte.
- Dem **unbekannten Ersteller** der Excel-Tabelle für die Zeiterfassung, welche von vielen Studenten genutzt wird.

Glossar

LaTeX	Bei LaTeX handelt es sich um ein spezielles Format für TeX. Unter einem Format versteht man in diesem Zusammenhang eine bestimmte Anzahl vordefinierter Kommandos, die beim Starten von TeX zur Verfügung stehen. Der Sinn dabei ist der, TeX so flexibel wie möglich zu halten. Dazu wurden nur sehr wenige Kommandos fest eingebaut, der Rest wird als Format dazu geladen.
TeX	Technisch gesehen handelt es sich bei TeX um einen Interpreter, der ca. 300 fest eingebaute Befehle kennt und einen komplexen Mechanismus zur Definition eigener Makros bereitstellt.
BFA	Behälter-Förderanlage
CSV	Comma Seperated Values
DOM	Document Object Model ist eine Programmierschnittstelle (API) für den Zugriff auf HTML- oder XML-Dokumente. Sie wird vom World Wide Web Consortium definiert.
ERP	Enterprise Resource Planning
Hibernate	Ein Open-Source-Persistenz Framework für Java
HOKO	Host-Kommunikation - Schnittstelle zur Middleware
HW	Hardware
IDOC	Intermediate Documents - Datencontainer, welche den Austausch von Business Daten zwischen einem SAP-System und einem anderen SAP oder nicht-SAP System erlauben
JDBC	Java Database Connectivity ist ein API der Java-Plattform, die eine einheitliche Schnittstelle zu Datenbanken verschiedener Hersteller bietet und speziell auf relationale Datenbanken ausgerichtet ist.
JUDE	Java and UML Developers Environment - UML Modeling Tool - [JUD04]
KTG	Kleinteil-Gerät
LAKOS	Lagerverwaltungs- und Kontroll-System der Stöcklin Software AG

LFV	LogFileVisualizer (lfv) - Die in diesem Dokument beschriebene Software.
LVS	Lagerverwaltungssystem
MF	Materialfluss
MFR	Materialflussrechner
PLC	Programmable Logic Controller, Englische Bezeichnung von SPS
PPS	Produktionsplanungssystem
RBG	Regalbediengerät
RegExp	Reguläre Ausdrücke (engl. regular expressions) dienen der Beschreibung von (Unter-)Mengen von Zeichenketten mit Hilfe syntaktischer Regeln.
RUP	Rational Unified Process. Ein Vorgehensmodell in der Softwareentwicklung.
SAP	Deutscher Softwarehersteller, SAP AG
SAX	Die Simple API for XML ist ein Pseudo-Standard, der ein Application Programming Interface (API) zum parsen von XML-Daten beschreibt.
SPS	Speicherprogrammierbare Steuerung
SQLite	SQLite ist eine Programmbibliothek, die ein relationales Datenbanksystem beinhaltet, welches die ACID-Eigenschaften erfüllt. Die Datenbank ist vor allem für den Embedded-Einsatz entworfen, für alle wichtigen Programmiersprachen existieren passende Datenbankschnittstellen. Durch die Entwicklung als Embedded lässt sich die Applikation direkt in entsprechende Anwendungen integrieren, so dass eine weitere Server-Software nicht benötigt wird.
TORD	Transport Order - SAP Transportauftrag
UML	**U**nified **M**odeling **L**anguage
VW	Verschiebewagen
Wiki	Wikis sind im World Wide Web verfügbare Seitensammlungen, die von den Benutzern nicht nur gelesen, sondern auch online geändert werden. Sie ähneln damit Content Management Systemen. Der Name stammt von wikiwiki, dem hawaiianischen Wort für <schnell>. Wie bei Hypertexten üblich, sind die einzelnen Seiten und Artikel eines Wikis durch Querverweise (Links) miteinander verbunden. `http://de.wikipedia.org/wiki/Wiki`

WMS	Warehouse Management System
XML	eXtensible Markup Language ist ein Standard zur Modellierung von halb-strukturierten Daten in Form einer Baumstruktur, der vom World Wide Web Consortium (W3C) definiert wird.

Literaturverzeichnis

[Fou05] FOUNDATION, The E. *Eclipse.org*. `http://www.eclipse.org/`. 2005

[Hel02] HELMUT, Kopka: *LaTeX Band 1: Einführung*. 3., überarbeitete Auflage. ADDISON-WESLEY, 2002

[JUD04] *JUDE - Java and UML Developers Environment*. `http://jude.change-vision.com/jude-web/`. 2004

[Koh04] KOHM, Markus. *Das KOMA-Script-Paket*. `http://www.komascript.de/`. 2004

[Lar04] LARMAN, Craig: *Applying UML and Patterns*. Third Edition. Prentice Hall PTR, 2004

[Owe06] OWENS, Michael: *The Definitive Guide to SQLite*. Apress, 2006