Masterarbeit

INBETRIEBNAHME-EMULATOR FÜR EIN WAREHOUSE-MANAGEMENT-SYSTEM

ausgeführt am

Fachhochschul-Masterstudiengang
Automatisierungstechnik-Wirtschaft

von
Ing. Ernest Maier, BSc.
1710322019

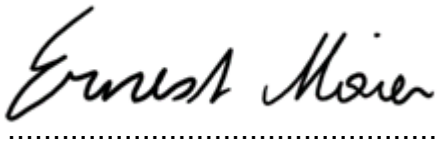
betreut und begutachtet von
FH-Prof. Dipl.-Ing. Dieter Lutzmayr

Graz, im Dezember 2018

.....
Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.



.....

Unterschrift

DANKSAGUNG

Ich möchte mich an dieser Stelle bei all jenen bedanken, die mich während der Erstellung dieser Arbeit sowie während des gesamten Studiums unterstützt und motiviert haben.

Ein ganz besonderer Dank gebührt meiner Freundin Iris Dorfegger, die mir während des Studiums Rückhalt gegeben hat und mir immer mit Rat und Tat zu Seite gestanden ist. Vor allem während Prüfungszeiten des Studiums und in den letzten Wochen vor der Abgabe dieser Arbeit hat sie mich moralisch unterstützt.

Ebenfalls möchte ich mich bei meinem Betreuer Dipl.-Ing. Dieter Lutzmayr für sein ausführliches und konstruktives Feedback während der gesamten Masterarbeit bedanken.

Abschließend gilt mein Dank auch meinen Eltern und Schwiegereltern in spe für ihre tatkräftige Unterstützung und ihr Verständnis, dass sie mich während der stressigen Phasen des Studiums nur wenig zu Gesicht bekommen haben.

KURZFASSUNG

Bei der vorliegenden Masterarbeit handelt es sich um ein Konzept für einen Warehouse-Management-System-Emulator, der für die Inbetriebnahme von Logistikanlagen entwickelt wird. Die Arbeit umfasst weiters die Umsetzung eines automatisierten Leistungsnachweises, der Testszenarien für die Fördertechnik und für Regalbediengeräte erstellen, vorbereiten und ausführen soll. Neben diesen Aufgaben zählen auch der Funktionsnachweis von Logistikanlagen und die allgemeine Unterstützung der Inbetriebnehmer bei ihrer Arbeit zu den Anforderungen an den Emulator. Für die Konzeptionierung wird das System in fünf Teile geteilt: die Datenbank, die Anwendung, die Benutzeroberfläche, den automatisierten Leistungstest und die Replay-Funktion. Hauptaugenmerk wird dabei auf die generische Entwicklung gelegt, wodurch sich der Emulator im Hochlauf selbständig konfiguriert und einfach erweitert werden kann.

ABSTRACT

This master's thesis deals with a concept of a warehouse management system (WMS) emulator, especially used for the commissioning of logistic plants. In addition, the thesis deals with the development of an automated performance testing function which aim is to define, prepare and run the test for the conveying system and the stacker cranes. For the development of the concept the system is divided into five subcomponents. These are the database, the user interface, the application, the performance testing function and the replay function. The main focus of the concept is the generic development of these five components, which should allow the emulator to be configured in the startup phase without making any changes in the software itself. The aim of this master's thesis is to develop a concept for a WMS emulator which has a modular design and to implement its performance testing function.

INHALTSVERZEICHNIS

1 Einleitung ... 1
2 Definition und Ziele ... 2
2.1 Ausgangssituation und Zielsetzung ... 2
2.2 Definition WMS ... 3
2.3 Zu emulierendes WMS ... 4
2.3.1 SAP-EWM ... 4
2.3.2 Jungheinrich-WMS ... 6
2.3.3 Vergleich Jungheinrich-WMS und SAP-EWM ... 7
2.4 Anforderungen an den Emulator ... 8
2.4.1 Funktionsnachweis ... 8
2.4.2 Leistungsnachweis ... 9
2.4.3 Replay-Funktion ... 9
2.5 Grenzen des Emulators ... 10
2.6 Zu behandelnde Funktionen ... 10
2.7 Leistungsnachweis von Logistik-Anlagen ... 10
2.7.1 Fördertechnik ... 11
2.7.2 Regalbediengeräte ... 13
3 Parametrierung des Emulators ... 16
3.1 Auszeichnungssprachen ... 16
3.1.1 Extensible Markup Language ... 16
3.1.2 JavaScript Object Notation ... 17
3.1.3 YAML Ain’t Markup Language ... 18
3.1.4 Evaluierungsergebnis ... 19
3.2 Automatische Generierung der Parameterdatei ... 19
4 Softwarearchitektur-Planung ... 20
4.1 Vorgehensmodell der Softwareentwicklung ... 20
4.2 V-Modell ... 21
4.3 Agile Softwareentwicklung ... 22
4.4 Hybrides Vorgehensmodell ... 23
4.5 Architekturmuster ... 24
4.5.1 Model View ViewModel (MVVM) ... 24
4.5.2 Object Relational Mapping (ORM) und das Entity Framework ... 25
4.6 Systemteil Definition ... 26
4.6.1 Datenbank ... 27
4.6.2 Anwendung ... 27
4.6.3 Grafische Benutzeroberfläche ... 27
4.6.4 Schnittstellendefinition der Systemteile ... 27
5 Datenbank ... 29
5.1 Anforderungen ... 29

5.2 Aufbau der Tabellen ... 30
5.3 Beziehungen zwischen den Tabellen ... 33
5.4 Microsoft Entity Framework ... 35
5.4.1 Early Prototyping ... 36
5.4.2 Probleme in der Verwendung ... 38
6 Anwendungssoftware ... 39
6.1 Architektur ... 39
6.2 Kommunikation zur Steuerungsebene ... 40
6.2.1 Anforderungen ... 40
6.2.2 Lösungskonzept ... 40
6.3 Meldepunkte ... 41
6.3.1 Anforderungen ... 41
6.3.2 Lösungskonzept ... 42
6.4 Routen / Lagerplatzsuche ... 45
6.4.1 Anforderungen ... 45
6.4.2 Lösungskonzept ... 45
6.5 Auslageraufträge ... 47
6.5.1 Anforderungen ... 47
6.5.2 Lösungskonzept ... 48
7 Benutzeroberfläche ... 49
7.1 Design ... 49
7.1.1 Hauptfenster ... 49
7.1.2 Begrüßungsbildschirm ... 50
7.1.3 Dialoge ... 50
7.2 Auszulösende Funktionen / anzuzeigende Informationen ... 51
7.3 Automatische Konfigurierung der Bedienoberfläche ... 52
7.4 Model View ViewModel in der praktischen Umsetzung ... 52
8 Automatisierte Leistungstests ... 57
8.1 Anforderungen ... 57
8.2 Lösungsweg ... 57
8.3 Lösungsweg Regalbediengerät ... 60
8.3.1 Definition ... 60
8.3.2 Vorbereitung ... 61
8.3.3 Messung ... 62
8.3.4 Vergleich ... 64
8.4 Lösungsweg Fördertechnik ... 65
8.4.1 Definition ... 65
8.4.2 Vorbereitung ... 66
8.4.3 Messung ... 66
8.5 Funktionstests ... 68
8.5.1 Leistungstest Regalbediengerät ... 68
8.5.2 Leistungstest Fördertechnik ... 69

9 Replay-Funktion 71
9.1 Anforderungen 71
9.2 Lösungskonzept 1 71
9.3 Lösungskonzept 2 73
9.4 Auswahl des Lösungskonzeptes 74
9.5 Schwierigkeiten des Konzepts in der praktischen Umsetzung 74
10 Ergebnisse und Ausblick 76
Literaturverzeichnis 78
Abbildungsverzeichnis 80
Tabellenverzeichnis 84
Abkürzungsverzeichnis 85

1 EINLEITUNG

Die vorliegende Masterarbeit behandelt die Entwicklung eines Warehouse-Management-System Emulators, der bei der Inbetriebnahme von Logistikanlagen zum Einsatz kommen soll. Sie umfasst die Erarbeitung eines entsprechenden Konzepts sowie die Umsetzung eines automatischen Leistungsnachweises für Fördertechnik und Regalbediengeräte.

Ziel des Emulators ist es, die Zeit, die für die Inbetriebnahme der Steuerungstechnik benötigt wird, zu verringern, indem die Inbetriebnehmer unabhängig von übergeordneten Systemen Automatik- und Leistungstests durchführen können. Unterstützt werden sie dabei durch speziell für die Inbetriebnahme entwickelte Funktionen, wie die Replay-Funktion.

Zu Beginn dieser Arbeit werden die nötigen Begriffe und Anforderungen an das zu entwickelnde System definiert. Weiters soll der Leistungsnachweis von Logistikanlagen erklärt werden, um die theoretischen Aspekte der geplanten Testfunktion zu erläutern. Ebenfalls Inhalt des theoretischen Teils sind die Parametrierung des Emulators, das verwendete Vorgehensmodell der Softwareentwicklung und die Erläuterung der verwendeten Architekturmuster.

Für die Entwicklung des Konzepts soll der Emulator in fünf Pakete geteilte werden: die Datenbank, die Benutzeroberfläche, die Anwendung, den Leistungstest und die Replay-Funktion. Zuerst werden für jede Komponente Anforderungen definiert, die die Basis für die folgenden Lösungskonzepte bilden. Zentraler Forschungsgegenstand dieses Teils der Masterarbeit ist die generische Entwicklung des Emulators, wodurch er möglichst erweiterbar und automatisch konfigurierbar werden soll.

Die derzeit händisch in der Steuerung programmierten Leistungstests sollen durch einfach konfigurierbare Tests, die im Emulator implementiert sind, abgelöst werden. Inbetriebnehmern wird es dadurch ermöglicht, diese neuen Leistungstests selbstständig zu konfigurieren, vorzubereiten, auszuführen und zu vergleichen, wodurch Zeit gespart wird und Schwachstellen schnell erkannt werden können.

Der abschließende Teil der Masterarbeit beschäftigt sich mit der Fehlersuche während Automatiktests. Vor allem für schwer bis nicht wiederholbare Störfälle soll eine Replay-Funktion im Emulator entwickelt werden, die es ermöglicht, den Fehler wiederholt darzustellen.

2 DEFINITION UND ZIELE

Im folgenden Kapitel werden die Ausgangssituation der Masterarbeit beleuchtet, relevante Begriffe und Systeme erklärt und die Ziele des Projektes erarbeitet.

Nach einer kurzen Beschreibung der Ausgangslage wird der Begriff „Warehouse-Management-System (WMS)“ erläutert und es werden die beiden zu emulierenden Systeme verglichen. In einem nächsten Schritt sollen die Anforderungen an den Emulator sowie die zu behandelnden Funktionen definiert werden. Zum Abschluss gibt das Kapitel einen allgemeinen Überblick über den Leistungsnachweis von Logistikanlagen, um ein detaillierteres theoretisches Verständnis für die Aufgaben des Emulators zu gewinnen.

2.1 Ausgangssituation und Zielsetzung

Die vorliegende Masterarbeit wird in Zusammenarbeit mit der Jungheinrich Systemlösungen GmbH (JSL) umgesetzt, die als Teil der Jungheinrich AG speziell in den Bereichen Steuerungs-, Lager- und Materialflusstechnik tätig ist.

Ziel der Arbeit ist es, die Inbetriebnahmezeit der von Jungheinrich errichteten Anlagen zu verkürzen und effizienter zu gestalten. Eine Möglichkeit, das zu erreichen, ist es, langwierige Automatiktests, bei denen Mitarbeiter der WMS- und Steuerungstechnik (ST)-Abteilung gemeinsam auf der Baustelle sein müssen, zu beschleunigen. Insbesondere dieser Teil der Inbetriebnahmephase gilt als besonders ressourcenaufwändig, da beide Seiten voneinander abhängig sind und sich durch Fehler gegenseitig aufhalten. Aktuell wird von Seiten der ST versucht, bereits vorab ohne Leitsystem die Anlage im Automatikbetrieb zu testen. Dafür müssen immer wieder Telegramme händisch erzeugt oder Daten der Ladeeinheiten händisch in die Datenbausteine der speicherprogrammierbaren Steuerung (SPS) eingetragen werden. Dieser Vorgang ist aufwändig und fehleranfällig – zusätzlich testet dieses händische Manipulieren nicht die gesamte Software, sondern nur kleine Teile davon.

Neben den langwierigen Automatiktests gestaltet sich auch der Leistungstest am Ende der Inbetriebnahmephase als zeitaufwändig und kompliziert, da er ohne WMS durchgeführt wird. Die verschiedenen benötigten Abläufe werden bis jetzt händisch in der Steuerung des Regalbediengeräts (RBG) programmiert. Durch den geplanten Emulator soll der Leistungstests über die Benutzeroberfläche gestartet werden können und danach automatisch ablaufen.

Die durchgeführten Automatiktests sowie der spätere Leistungstest sollen durch die zu entwickelnde Emulationssoftware insgesamt schneller und genauer umgesetzt werden. Dieses Parallel-Engineering von Steuerungstechnik und WMS soll, wie in Abbildung 1 ersichtlich, die zuvor angesprochene Testphase von 3 bis 6 Wochen auf 1 bis 2 Wochen verkürzen.

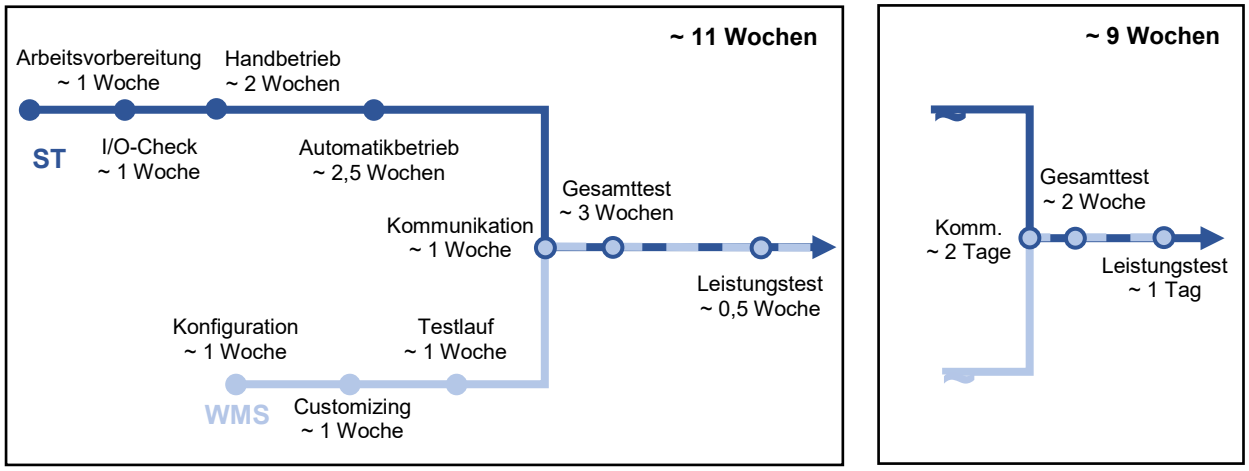


Abbildung 1: Zeitlicher Vergleich der Inbetriebnahmephase mit und ohne WMS Emulator, Quelle: Eigene Darstellung

Der Emulator soll erstmals im November 2019 bei einem Großprojekt des Discounter-Konzerns Lidl im Norden von Wien zum Einsatz kommen. Das zu emulierende WMS ist das SAP-Extended Warehouse Management (EWM). Die Inbetriebnahme des Systems wird dabei von SAP selbst durchgeführt. Diese erfolgt aber erst nach dem Funktions- bzw. Leistungsnachweis der Steuerungsebene, die von Jungheinrich Systemlösungen in Betrieb genommenen wird. Die terminlichen Vorgaben des Kunden machen den Einsatz des Emulators notwendig.

2.2 Definition WMS

Ein Warehouse-Management-System (WMS) ist ein softwarebasiertes System, das den Wareneingang, die Stellplätze innerhalb eines Lagers und den Versand von Gütern steuert. Das WMS bestimmt Quelle und Ziel der Güter im Lager und verwaltet Aufträge. Nach neuestem technischen Stand sind Warehouse-Management-Systeme modular aufgebaut, wodurch kundenspezifische Lösungen schnell und kostengünstig realisiert werden können. Durch die Integration von Warehouse-Management-Systemen in die bestehende Systemlandschaft eines Unternehmens ergeben sich Schnittstellen zur Kommunikation mit anderen Systemen. Übergeordnete Systeme sind Enterprise-Resource-Planning (ERP)-Systeme, die Planung und Organisation von Aufgaben übernehmen, um vorhandene Ressourcen bestmöglich einzuteilen. Als hierarchisch untergeordnetes System gilt die Steuerungsebene mit Materialflusssteuerungssystemen (MFS) oder speicherprogrammierbaren Steuerungen (SPS), die direkt Lagerfahrzeuge, Fördersysteme, Sorter und andere Lagereinrichtungen steuern. Die Kern- und Zusatzfunktionen eines WMS sowie die Schnittstellen zu über- und untergeordneten Systemen können der Abbildung 2 entnommen werden. [1]

[1] Vgl. Geißen/Bodden-Streubühr/Geldmacher/Ludwigs/Pfisterer/Pott/Spee (2014), S. 5.

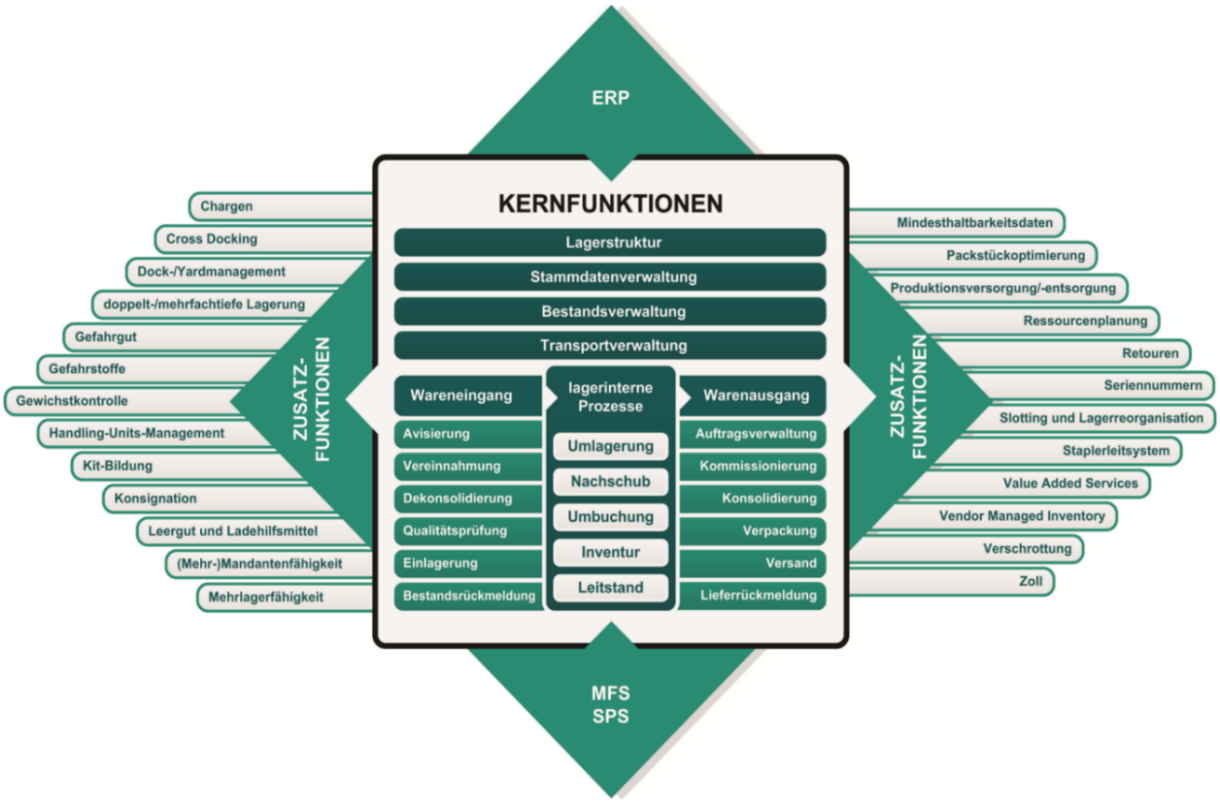


Abbildung 2: Kern- und Zusatzfunktionen eines WMS, Quelle: VDI 3601, S. 6.

2.3 Zu emulierendes WMS

Die zu entwickelnde Softwarelösung soll zwei verschiedene Systeme emulieren. Zum einen das von Jungheinrich selbst angebotene Jungheinrich-WMS und zum anderen das für das Großprojekt mit Lidl benötigte SAP-Extended Warehouse Management (EWM). Beide Systeme sollen in den folgenden Kapiteln kurz erläutert und im Anschluss gegenübergestellt werden.

2.3.1 SAP-EWM

SAP-Extended Warehouse Management ist seit 2006 Bestandteil des SAP Supply Chain Management (SCM) und unterstützt bei sämtlichen Aufgaben in der Logistikkette. Die vollständige Integration in das SAP-Enterprise-Resource-Planning-System ermöglicht es, Warenbewegungen, die von Geschäftsprozessen in anderen Bereichen angestoßen wurden, zu überwachen und zu steuern. Automatisierte Lagerumgebungen können ohne den Einsatz von Materialflusssystemen eines Drittanbieters zur Gänze überwacht und gesteuert werden. Die Eingliederung des SAP-EWM-Systems in bestehende SAP-Systeme wird in Abbildung 3 ersichtlich.[2]

[2] Vgl. SAPAG (Hrsg.) (2010), S. 4 f.

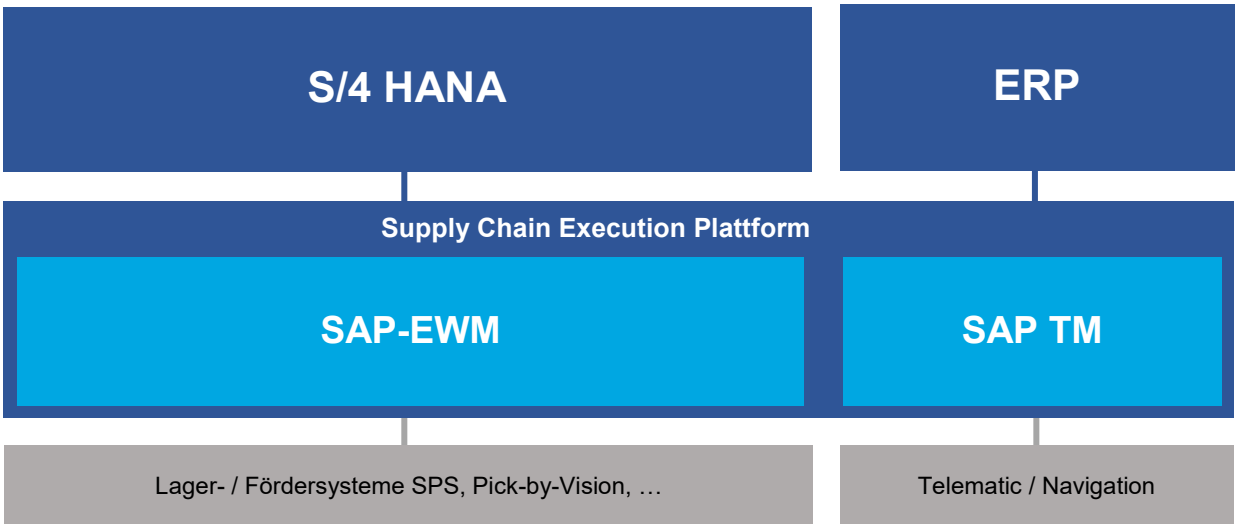


Abbildung 3: Eigliederung des SAP-EWM in die SAP Systemlandschaft, Quelle: Eigene Darstellung

Als Kernfunktionen des SAP-EWM-Systems gelten folgende Funktionen:[3]

Wareneingang:

- *Yard Management:*
  Das Steuern und Organisieren aller Bewegungen auf dem Frachthof.
- *Ein- und Auslagerstrategien:*
  Das Bestimmen von optimalen Lagerplätzen für die Einlagerung und Kommissionierung.
- *Erwarteter Wareneingang:*
  Einkaufsbelege des ERP-Systems werden als erwarteter Wareneingang dargestellt.
- *Cross-Docking:*
  Das Weiterleiten von Handling Units (HU) vom Wareneingang direkt zum Warenausgang, um die Zahl der bearbeiteten HUs zu erhöhen.
- *Logistische Zusatzleistungen:*
  Zu diesen Leistungen zählen etwa Montage, Etikettierung, Verpackung und Kitting.
- *Qualitätsmanagement (QM):*
  Das QM wird durch die Quality Inspection Engine (QIE) unterstützt.

Warenausgang:

- *Wellenmanagement:*
  Das Bilden von Arbeitspakten aus einzelnen Kommissionierungsaktivitäten und Umbuchungen.
- *Nachschubsteuerung:*
  Die bedarfsgemäße Steuerung des Bestandes in ihren Kommissionierbereichen.

[3] Vgl. SAPAG (Hrsg.) (2010), S. 6 f.

- *Produktionsversorgung:*
  Das Beliefern des Produktionsversorgungsbereiches, damit Produkte und Werkzeuge für diverse Fertigungsaufträge verwendet werden können.

Lagerverwaltung / -optimierung:

- *Handling Unit Management:*
  Als HU wird eine Einheit aus Packmittel (Ladungsträger und Verpackung) und verpacktem Produkt bezeichnet. Sie stellt alle produktrelevanten Informationen bereit.
- *Lagerungsdisposition:*
  Das Ermitteln des optimalen Lagerplatzes unter Berücksichtigung von Produkt-, Bedarfs- und Verpackungsdaten.
- *Inventur:*
  Die Gewährleistung, dass sich das richtige Produkt in der richtigen Menge an der richtigen Stelle im Lager befindet.

Distributionszentrum:

- *Planung und Überwachung:*
  Der Lagerverwaltungsmonitor des SAP-EWMs bietet dem Benutzer die Möglichkeit, sämtliche Funktionen innerhalb des Lagers zu steuern und bietet ihm einen Überblick über alle Lageraktivitäten.
- *Datenfunkanbindung:*
  Das Erfassen von Daten über Barcodes oder Tags.
- *Ressourcenmanagement:*
  Das Zuordnen von Ressourcen zu Queues sowie das Zuteilen von Lageraufträgen an Lagermitarbeiter.
- *Arbeitsmanagement:*
  Die Planung und Steuerung der Arbeitseinsätze von Mitarbeitern und die Messung ihrer Leistung.
- *Automatisierung und Unterstützung von Materialflusssystemen:*
  Über Schnittstellen zu Lagerfahrzeugen, Fördersystemen und anderem Equipment können alle Produktbewegungen in die Lagerverwaltung aufgenommen werden.

2.3.2 Jungheinrich-WMS

Das Jungheinrich-WMS ist das von Jungheinrich Systemlösungen entwickelte Warehouse-Management-System und dient der Verwaltung, Steuerung und Optimierung von Lagern. Es kann sowohl für manuelle als auch für teil- und vollautomatisierte Lager eingesetzt werden.

Ähnlich wie das SAP-EWM-System besteht auch das Jungheinrich-WMS aus Basis und Zusatzfunktionen. Das Jungheinrich-WMS ist dabei modular aufgebaut und besteht aus einer Vielzahl von einzelnen Bausteinen, die je nach Bedarf aktiviert und deaktiviert werden können. Die Basismodule sind in der Abbildung 4 aufgelistet.

Abbildung 4: Basismodule Jungheinrich-WMS, Quelle: Eigene Darstellung

Jungheinrich bietet im Unterschied zu SAP kein eigenes ERP-System an. Es besteht jedoch die Möglichkeit, das Jungheinrich-WMS an ein beliebiges ERP-System zu koppeln. Die Schnittstelle dafür ist klar definiert und kann auf Wunsch des Kunden auch spezifisch an das vom Unternehmen eingesetzte ERP-System angepasst werden. Im Bereich der Steuerungsebene besteht auch die Möglichkeit, Steuerungen verschiedenster Unternehmen anzubinden.

2.3.3 Vergleich Jungheinrich-WMS und SAP-EWM

Beide Warehouse-Management-Systeme haben ihre Vor- und Nachteile – welches die bessere Wahl darstellt, hängt immer von der Anlage und ihren Anforderungen ab. Inhalt dieses Kapitels ist es, die Systeme gegenüberzustellen und die Stärken und Schwächen beider kurz aufzuzeigen.

Jungheinrich-WMS	SAP-EWM
Vorteile: • Anbindung an unterschiedliche ERP-Systeme • Moderne Bedienoberfläche • Modularität • Vollständige Vorwärtskompatibilität trotz Customizing	Vorteile: • Reduktion der Schnittstellen durch die Integration in die Gesamtlösungslandschaft • Synergieeffekte mit SAP-ERP • Einheitliche Benutzeroberfläche
Nachteile: • Schnittstelle zu externem ERP-System notwendig	Nachteile: • Sehr hohe Anschaffungskosten • Vorteile können nur in Verbindung mit der Gesamtlösung SAP-ERP genützt werden • Geringe Flexibilität durch Standardsystem

Tabelle 1: Vergleich Jungheinrich-WMS und SAP-EWM, Quelle: Eigene Darstellung

Die Stärken des SAP-Extended Warehouse Management liegen in Anlagen, in denen bereits andere SAP Produkte zum Einsatz kommen. Ohne die Synergieeffekte mit anderen Systemen wie zum Beispiel SAP-ERP überwiegen die Nachteile wie die hohen Anschaffungskosten und die geringe Flexibilität. Die Stärken des Jungheinrich-WMS liegen dagegen in der Flexibilität, der Modularität und der modernen Benutzeroberfläche.

2.4 Anforderungen an den Emulator

Unter dem Begriff Emulator versteht man das Nachahmen bestimmter Funktionen einer Software oder eines Systems. Ziel des geplanten Emulators ist es, Warehouse-Management-Systeme verschiedener Anbieter nachbilden zu können, um die Inbetriebnahme der Steuerungsebene zu erleichtern und die Funktion sowie ihre Leistung nachzuweisen. In den folgenden Kapiteln werden die Anforderungen an das System für die genauere Spezifikation in folgende drei Teile gegliedert: Funktionsnachweis, Leistungsnachweis und Replay-Funktion.

2.4.1 Funktionsnachweis

Das Betreiben der Anlage im Automatikbetrieb während und am Ende der Inbetriebnahmephase soll dem Kunden als Funktionsnachweis dienen. Dabei muss darauf geachtet werden, dass die Kommunikation zwischen SPS und dem WMS Emulator identisch mit dem realen WMS funktioniert. Da es möglich sein soll, verschiedene WMS zu emulieren, soll der Kommunikationsteil des Emulators vollständig abgekapselt vom restlichen Emulator funktionieren. Durch diese Trennung wird das Emulieren von verschiedenen Leitsystemen einfach gestaltet.

Um neue Ladeeinheiten am Beginn der Logistikanlage zu registrieren, soll eine Wareneingangsfunktion geschaffen werden, die neben der Registrierung auch einfache Prüfungen durchführt. Ein Beispiel dafür wäre die Kontrolle der Ladeeinheitennummer (LE-Nummer), bei der geprüft wird, ob die Nummer bereits im System vorhanden ist oder nicht.

Neben dem Anlegen neuer Ladeeinheiten im System soll es auch möglich sein, diese am Ende der Anlage an definierten Warenausgangsstationen wieder zu entfernen. Dieser Vorgang kann auch an jedem anderen Platz der Anlage für etwaige Tests manuell durchgeführt werden.

Damit die Ware in der Anlage ihr vorgesehenes Ziel erreicht, soll das System Möglichkeiten haben, Routen zu bilden. Eine Route verknüpft dabei mehrere Zwischenziele zu einem Pfad, über den die Ware an das gewünschte Endziel gelangt. Bei Bedarf soll der Emulator vordefinierte Routen auch neu berechnen können, wenn das Zwischenziel zum Beispiel nicht mehr erreichbar ist. Bei der Berechnung der Route ist zu berücksichtigen, dass die Wegabschnitte zwischen einzelnen Zielen verschiedene Eigenschaften haben können. Beispiele dafür wären verschiedene Höhenklassen oder eine maximale Anzahl an Ladeeinheiten.

Da es bei der Inbetriebnahme vorkommen kann, dass einzelne Anlagenteile noch keinen Automatikbetrieb zulassen, während andere bereits fertiggestellt wurden, soll es dem Bediener möglich sein, einzelne Wegabschnitte der Anlage zu sperren.

2.4.2 Leistungsnachweis

Beim Verkauf einer Logistikanlage werden immer auch Leistungen definiert, die vor Übernahme der Anlage durch den Kunden geprüft werden. Für deren Nachweis wird versucht, an definierten Eckpunkten der Anlage die vertraglich vereinbarten Leistungen nachzustellen und zu überprüfen. Der WMS-Emulator soll dabei maßgeblich unterstützen und es ermöglichen, Leistungstestszenarien zu definieren. Diese Szenarien unterscheiden sich bei Regalbediengeräten und Fördertechnik. Im Bereich des Hochregallagers sollen dabei Lastspiele definiert bzw. automatisch erstellt werden können, die den Fédération Européenne de la Manutention (FEM)-Richtlinien entsprechen. Im Bereich der Fördertechnik sollen Leistungen an bestimmten Eckpunkten oder Wegabschnitten vorgegeben werden können, die während des Tests möglichst genau eingehalten werden. Dies ermöglicht eine präzise Aufteilung der Leistung innerhalb der Anlage.

Neben der Definition der Szenarien sollen auch eine Datenauswertung und ein Vergleich zwischen mehreren Szenarien möglich sein.

2.4.3 Replay-Funktion

Um die Qualität der in Betrieb genommenen Anlage zu erhöhen, soll eine Replay-Funktion implementiert werden. Sie gibt dem Inbetriebnehmer die Möglichkeit, nach einer Störung den Zustand der Anlage im WMS einzufrieren. Danach wird zunächst eine Fehlersuche mittels Aufschlüsselung aller Bewegungen der letzten 10 Minuten ermöglicht. Nach beendeter Suche kann der Inbetriebnehmer die letzten Bewegungen rückgängig machen. Dabei ist es notwendig, dass die Ladeinheiten physikalisch ebenfalls auf die ursprüngliche Position zurückgebracht und die Bewegungen in der exakt gleichen Abfolge wieder abgespielt werden können. Die Replay-Funktion soll dem Inbetriebnehmer dabei helfen, potentielle Fehlerquellen schnell zu erkennen und zu beheben. Von dieser Funktion ausgenommen sind Positionen,

deren Bewegungen nicht rückgängig gemacht werden können, zum Beispiel Gefällestrecken. Andere Sonderelemente der Fördertechnik, die wiederholt durchlaufen werden können, wie etwa Labelmaschinen und Umreifungsmaschinen, sollen in die Replay-Funktion integriert werden.

2.5 Grenzen des Emulators

Der zu entwickelnde Simulator soll – auch wenn er die Basisfunktionalitäten eines WMS abdecken könnte – nie als reales Leitsystem zum Einsatz kommen. Auch soll der Emulator nicht mit dem Jungheinrich-WMS in Verbindung gebracht werden, da er es weder ersetzen kann noch für dieses Einsatzgebiet gedacht ist. Der Emulator ist nur für den internen Gebrauch und die Bedienung durch Jungheinrich Systemlösungen-Mitarbeiter bestimmt und soll keinem Kunden zur Eigennutzung zur Verfügung gestellt werden.

Die Leistung, sowie die Verfügbarkeit des Emulators müssen nicht zwingend die eines realen Leitsystems erreichen. Weiters wird der Datensicherheit nur ein geringes Maß an Bedeutung zugerechnet, da das Ziel des Emulators lediglich der funktionale Betrieb sowie die Unterstützung bei der Inbetriebnahme der Steuerungsebenen von Logistikanlagen im Automatikbetrieb ist.

2.6 Zu behandelnde Funktionen

Der große Aufwand, der für die Entwicklung des Emulators angenommen wird, macht es notwendig Entwicklungsschwerpunkte zu setzen. Diese Schwerpunkte sollen besonders interessant für die genauere Betrachtung im Rahmen einer wissenschaftlichen Arbeit sein und werden in diesem Kapitel definiert.

Als besonders erwähnenswert gelten das Deklarieren, Ausführen und Vergleichen von Leistungstestszenarien sowie die Umsetzung der Replay-Funktion. Da diese Funktionen aber von Basisfunktionen abhängig sind und kaum eigenständig getestet werden können, sollen auch Teile der Basisfunktionen im praktischen Teil dieser Arbeit behandelt werden. Zu diesen zählen folgende Aufgaben:

- Das Verbinden der Applikation mit einer Datenbank sowie das Lesen, Hinzufügen, Entfernen und Ändern von Einträgen in dieser.
- Das Anlegen und Bearbeiten von Einlager- und Auslageraufträgen im Hochregallager.
- Das Bilden von Routen in der Fördertechnik sowie die automatische Platzsuche im Hochregallager.

Zusätzlich zu diesen Funktionen soll die graphische Benutzeroberfläche, die die Interaktion mit dem Emulator möglich macht, in einem eigenen Kapitel behandelt werden. Dabei wird speziell auf die Verbindung zwischen dem User Interface (UI) und der Anwendung eingegangen.

2.7 Leistungsnachweis von Logistik-Anlagen

Ein wichtiger Teil des Emulators und damit auch dieser Arbeit ist der Leistungsnachweis von Fördertechnik und Regalbediengeräten. Um die theoretischen Aspekte dieses Themas besser zu verstehen, werden sie innerhalb dieses Kapitels kurz erläutert. Dabei soll erklärt werden, was Leistung im Bereich der Fördertechnik und des Hochregallagers bedeutet und wie diese bestimmt werden kann. Zusätzlich wird ein kleiner Einblick in die Materialflussrechnung gegeben.

2.7.1 Fördertechnik

Den Großteil heutiger Materialflusssysteme stellt ein Netzwerk aus seriell und parallel angeordneten Knoten dar, wobei der Grad der Flexibilität des Systems die Komplexität des Netzwerks beeinflusst. Je flexibler das System, desto höher die zu bildende Vernetzung zwischen den einzelnen Knoten. Diese werden auch als Elementarstationen bezeichnet und teilen sich auf in Quellen, Senken, Verteiler, Bearbeitungsstellen, Knoten und Zusammenführungen. [4]

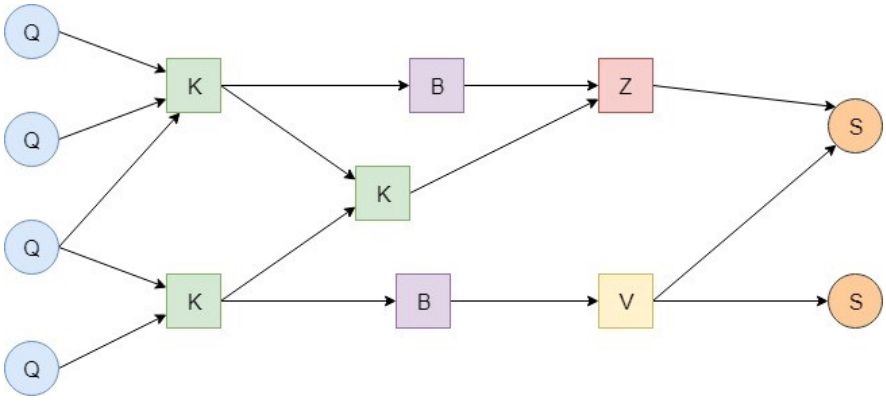


Abbildung 5: Materialflussschema, Quelle: Eigene Darstellung.

Der Abbildung 5 kann die schematische Zusammensetzung eines solchen Materialflusssystems mit den verschiedenen Knoten entnommen werden.

Die zuvor erwähnten Elementarstationen eines Fördersystems unterscheiden sich in ihren mechanischen Elementen nur sehr gering und werden weiterführend als Grundelemente der Materialflussrechnung bezeichnet.[5]

Förderstrecken:

Dieses Element ermöglicht den unbehinderten Transport von Ladeeinheiten (LE) zwischen einer Quelle und einer Senke. Der Vorgang wird in Abbildung 6 grafisch dargestellt.[6]

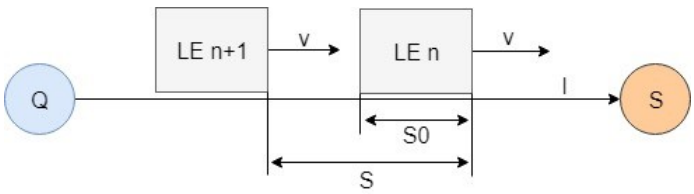


Abbildung 6: Ladeeinheiten auf einer Förderstrecke der Länge l, Quelle: Eigene Darstellung

Für die Berechnung des Durchsatzes ist die Kenntnis über das Geschwindigkeit-Zeit-Verhalten von Bedeutung, wobei zwischen aktiven und passiven Förderstrecken unterschieden wird. Aktive

[4] Vgl. Dieter/Furmanns (2009), S. 4 f.

[5] Vgl. Dieter/Furmanns (2009), S. 8.

[6] Vgl. Dieter/Furmanns (2009), S. 11.

Förderstrecken wie Rollen- oder Gurtförderer weisen in der Regel eine konstante Fördergeschwindigkeit auf, passive wie Fahrwege und Schienen nicht. Beschleunigungs- und Bremsvorgänge dieser Förderstrecken werden unter Annahme einer mittleren konstanten Geschwindigkeit vereinfacht.[7]

Der Durchsatz λ der Förderstrecke berechnet sich damit aus der Geschwindigkeit v und dem Abstand der Ladeeinheiten zueinander s . Der maximale Durchsatz γ, auch als Grenzdurchsatz bezeichnet, kann in der Praxis nicht erreicht werden und ergibt sich, wenn die Ladeinheiten lückenlos aneinandergereiht sind. Mittels Durchsatz und Grenzdurchsatz lässt sich auch der Auslastungsgrad ρ des Förderelementes bestimmen. Ist dieser Wert kleiner als 1, so gilt die Förderstrecke als nicht vollständig ausgelastet.[8]

$$\lambda = \frac{v}{s} \quad \gamma = \frac{v}{s_0} \quad \lambda \leq \gamma \quad \rho = \frac{\lambda}{\gamma} \leq 1 \qquad (2.1)$$

$\lambda / \frac{1}{\mathrm{s}}$	Durchsatz
$v / \frac{\mathrm{m}}{\mathrm{s}}$	Geschwindigkeit
s / m	Weg
$\gamma / \frac{1}{\mathrm{s}}$	Grenzdurchsatz
$\rho / 1$	Auslastungsgrad

Teilstetige Verzweigungen für zwei Richtungen:

Als teilstetig werden Verzweigungen bezeichnet, wenn in einige Richtungen eine stetige Abfertigung möglich ist und in andere nicht. In Abbildung 7 ist ersichtlich, dass der Transportweg zwischen Q und S1 als stetig angenommen werden kann und der Weg zwischen Q und S2 als unstetig.[8]

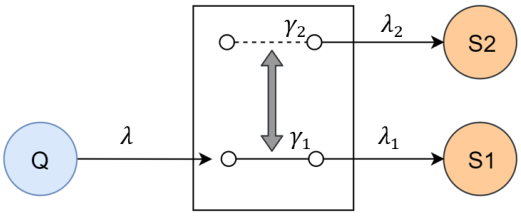


Abbildung 7: Teilstetig Verzweigung für 2 Richtungen, Quelle: Eigene Darstellung

Um den Durchsatz dieses Grundelements zu berechnen, muss neben der Durchsatzberechnung der Förderstrecke selbst auch auf den Wechsel zwischen den Senken eingegangen werden. Dieser Umschaltvorgang wird mittels partiellem Grenzdurchsatz γ_1 und γ_2 berücksichtigt, der sich wie in Formel 2**Fehler! Verweisquelle konnte nicht gefunden werden.**.2 bestimmen lässt. Die dafür benötigten Z eiten Δt_1 und Δt_2 ergeben sich aus der Zeit, die eine Ladeeinheit zwischen dem Eintritt und dem Austritt aus der Verzweigung benötigt. Abbildung 7 lässt dabei erkennen, dass die Zeit Δt_2, die für die Hin- und Rückfahrt benötigt wird, weitaus größer ist als die Zeit Δt_1.[9]

$$\gamma_1 = \frac{1}{\Delta t_1} \quad \gamma_2 = \frac{1}{\Delta t_2} \quad \gamma_1 \gg \gamma_2 \qquad (2.2)$$

t / s	Zeit

[7] Vgl. Dieter/Furmanns (2009), S. 11 ff.

[8] Vgl. Dieter/Furmanns (2009), S. 23.

[9] Vgl. Dieter/Furmanns (2009), S. 26 f.

Der betriebliche Durchsatz λ der Verzweigung ergibt sich aus der Summe der Einzeldurchsätze λ_1 und λ_2, wobei sich dieser je nach Aufteilung auf die Senke 1 und 2 ändern kann. Die Auslastung der gesamten Verzweigung bildet sich ähnlich wie der Durchsatz aus den Einzelauslastungen und darf auch hier den Wert 1 nicht überschreiten.[10]

$$\rho_1 = \frac{\lambda_1}{\gamma_1} \leq 1 \quad \rho_2 = \frac{\lambda_2}{\gamma_2} \leq 1 \quad \rho = \rho_1 + \rho_2 \leq 1 \qquad (2.3)$$

2.7.2 Regalbediengeräte

Die Leistung, die bei Regalbediengeräten (RBG) nachzuweisen ist, wird meist als Umschlagsleistung bezeichnet. Unter ihr versteht man die Anzahl an Einlagerungen und/oder Auslagerungen innerhalb einer definierten Zeit. Sie ist abhängig von:[11]

- Dem Bewegungsablauf des RBGs
- Der Anordnung der Ein- und Auslagerpunkte
- Der Geräteleistung

Neben der Umschlagsleistung sind die einzelnen Spielzeiten des Regalbediengerätes von Bedeutung. Sie geben an, wie lange das RBG für eine Einlagerung, Auslagerung oder ein kombiniertes Ein- und Auslagerspiel benötigt. Die Spielzeit besteht sowohl aus produktiven als auch aus unproduktiven Zeitanteilen. Als unproduktiv werden dabei Bewegungen des RBGs ohne Ware – sogenannte Leerfahrten – bezeichnet. Diese können durch das Kombinieren von Ein- und Auslagerung verringert werden. In Abbildung 8 wird der Aufbau der verschiedenen Arbeitsspiele dargestellt. Leerfahrten sind in Rot gekennzeichnet.[12]

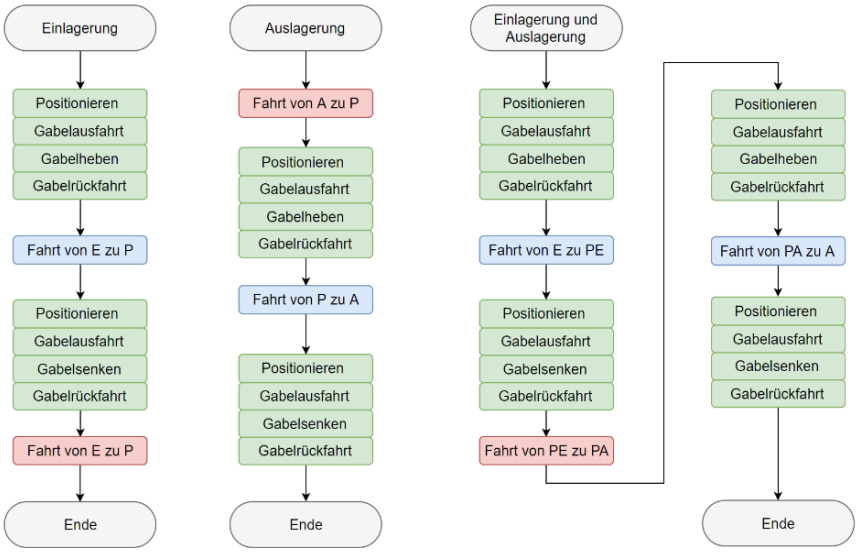


Abbildung 8: Aufbau des Arbeitsspieles eines Regalbediengerätes, Quelle: Eigene Darstellung

[10] Vgl. Dieter/Furmanns (2009), S. 24.

[11] Vgl. Federation Europeenne de la Manutention (Hrsg.) (2003), S. 3.

[12] Vgl. Dieter/Furmanns (2009), S. 197.

Zur Berechnung der Spielzeit müssen alle konstanten Zeitwerte und veränderlichen Wegzeiten des Arbeitsspiels summiert werden. Die Zeiten hängen dabei von den technischen Daten des Gerätes und der Wegstrecke in X- und Y-Richtung ab. Um ein repräsentatives Ergebnis der Spielzeit zu erhalten, muss die veränderliche Wegzeit gemittelt werden. Da diese durchschnittliche Spielzeitberechnung sehr komplex und aufwändig ist, wird in diesem Kapitel nur auf die in der FEM-Norm beschriebene praxisnahe Annäherung eingegangen. Bedeutend für diese Annäherung ist die Position der Ein- und Auslagerung innerhalb des Lagers. Dabei werden sechs verschiedene Fälle unterschieden:[13]

- Fall 1: Ein- und Auslagerung am unteren Eckpunkt
- Fall 2: Einlagerung am linken Eckpunkt, Auslagerung am rechten Eckpunkt
- Fall 3: Ein- und Auslagerung sind sich gegenüber, angehoben im Eckpunkt
- Fall 4: Ein- und Auslagerung sind gleichermaßen in X-Richtung verschoben
- Fall 5: Einlagerung ist am Eckpunkt, die Auslagerung ist in Y-Richtung angehoben
- Fall 6: Einlagerung ist in Y-Richtung angehoben, Auslagerung befindet sich am Eckpunkt

Da die Fälle 1 und 5 bei Hochregallagern von Jungheinrich am häufigsten vorkommen, soll auf diese weiterführend genauer eingegangen werden.

Fall 1: Ein- und Auslagerung am unteren Eckpunkt

In diesem Fall liegen die Ein- und Auslagerung im linken unteren Eckpunkt des Lagers, die Koordinaten der Aufnahme- bzw. Abgabepunkte können wie in Formel 2.4 mit Hilfe der maximalen Lagerkoordinaten berechnet werden.

$$P1_X = \frac{1}{5} * L \qquad P1_Y = \frac{2}{3} * H \qquad (2.4)$$
$$P2_X = \frac{2}{3} * L \qquad P2_Y = \frac{1}{5} * H$$

$P1_x/1$ X-Koordinate Punkt 1
$P1_y/1$ Y-Koordinate Punkt 1
$P2_x/1$ X-Koordinate Punkt 2
$P2_y/1$ Y-Koordinate Punkt 3
$L/1$ Anzahl an X-Koordinaten
$H/1$ Anzahl an Y-Koordinaten

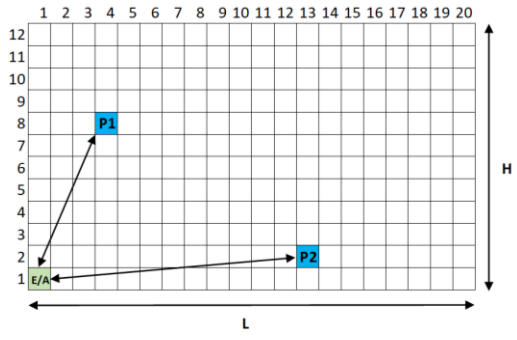


Abbildung 9: Mittleres Einzelspiel Fall 1,
Quelle: Eigene Darstellung

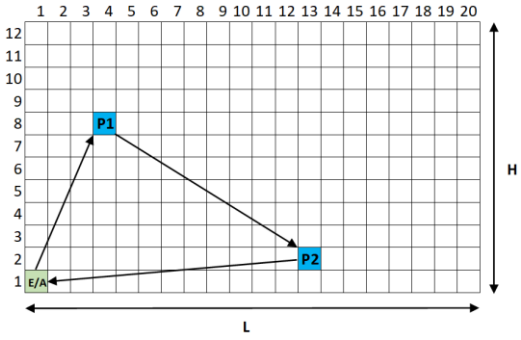


Abbildung 10: Mittleres Doppelspiel Fall 1,
Quelle: Eigene Darstellung

[13] Vgl. Federation Europeenne de la Manutention (Hrsg.) (2003), S. 6.

$$t_{m1} = \frac{1}{2} * (t_{E\,P1} + t_{A\,P2}) + t_{Constant} \quad (2.5)$$

t_{m1}/s	Mittlere Zeit eines Einzelspiels
$t_{E\,P1}/\mathrm{s}$	Fahrzeit zwischen der Einlagerung und P1
$t_{E\,P1}/\mathrm{s}$	Fahrzeit zwischen der Auslagerung und P2
$t_{Constant}/\mathrm{s}$	Von der Wegstrecke unabhängige Zeit

$$t_{m2} = (t_{E\,P1} + t_{P1\,P2} + t_{P2\,A}) + t_{Constant} \quad (2.6)$$

t_{m2}/s	Mittlere Zeit eines Doppelspiels
$t_{P1\,P2}/\mathrm{s}$	Fahrzeit von P1 zu P2

Fall 5: Einlagerung ist am Eckpunkt, die Auslagerung ist in Y-Richtung angehoben

In diesem Fall liegt die Einlagerung im linken unteren Eckpunkt des Lagers und die Auslagerung ist um Y_A in Y-Richtung verschoben, die Koordinaten der Aufnahme- bzw. Abgabepunkte können wie in Formel 2.7 mit Hilfe der maximalen Lagerkoordinaten und Y_A berechnet werden.

$$P1_x = \frac{1}{5} * L \qquad P1_Y = \frac{2}{3} * H + \frac{1}{6} * Y_A \quad (2.7)$$

$$P2_X = \frac{2}{3} * L \qquad P2_Y = \frac{1}{5} * H + \frac{1}{6} * Y_A$$

$Y_A/1$ Abstand zwischen Einlagerung und Auslagerung in Y-Richtung

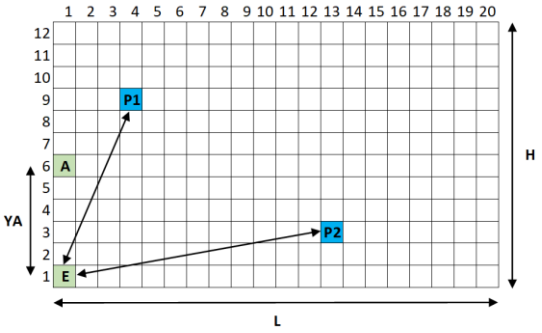


Abbildung 11: Mittleres Einzelspiel Einlagerung Fall 5, Quelle: Eigene Darstellung

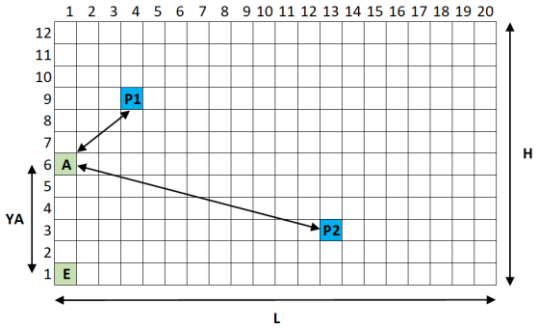


Abbildung 12: Mittleres Einzelspiel Auslagerung Fall 5, Quelle: Eigene Darstellung

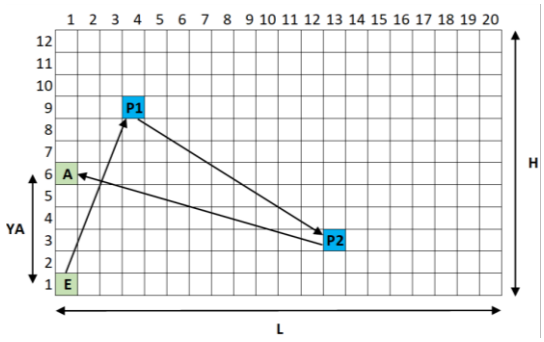


Abbildung 13: Mittleres Doppelspiel Fall 5, Quelle: Eigene Darstellung

3 PARAMETRIERUNG DES EMULATORS

Um den Emulator zukünftig für verschiedene Projekte mit unterschiedlichen Anforderungen einsetzen zu können, soll es die Möglichkeit geben, ihn projektspezifisch zu konfigurieren. Änderungen in der Software des Emulators sollen dafür nicht nötig sein.

Im folgenden Kapitel sollen die verschiedenen Parameterdateien-Formate gegenübergestellt und verglichen werden, die ein Parametrieren des Emulators ermöglichen sollen. Am Ende des Kapitels wird zusätzlich die Generierung einer solchen Parameterdatei behandelt.

3.1 Auszeichnungssprachen

Neben der wohl bekanntesten Auszeichnungssprache „Extensible Markup Language (XML)“ gibt es eine Vielzahl anderer Sprachen, die verschiedene Vorteile gegenüber XML vorweisen. Besonders hervorzuheben sind dabei die Sprachen „JavaScript Object Notation (JSON)“ und „YAML Ain’t Markup Language (YAML)“, die aktuell weltweit immer mehr an Bedeutung gewinnen.

Um für diese Arbeit die geeignete Auszeichnungssprache zu finden, soll eine Nutzwertanalyse erstellt werden. Die Gewichtung der Kriterien erfolgt subjektiv durch den Entscheider mittels der „Direct Rating Methode“. Die Direct Rating Methode ist eine einfache Möglichkeit, verschiedene Kriterien nach ihrer Relevanz zu ordnen und die Entscheidungsfindung transparent zu halten. Die Relevanz wird dabei meist in zehn Stufen (von 1-10) unterteilt. Nachdem jedem Kriterium eine Relevanzstufe zugeschrieben wurde, muss diese durch die Summe aller Gewichtungen dividiert und als Prozentzahl angegeben werden. Die Beurteilung der Kriterien wird, wie die Gewichtung, in Stufen von eins bis zehn vorgenommen, wobei die Beurteilung eins am schlechtesten und zehn am besten ist.[14]

Die Beurteilung der Kriterien erfolgten durch eine Befragung der Entscheidungs- und Erfahrungsträger von Jungheinrich Systemlösungen.

3.1.1 Extensible Markup Language

XML dient zur Definition anderer Auszeichnungssprachen, die Textdokumente, Vektorgraphiken oder andere Daten beschreiben. Sie ist ein einfaches textbasiertes Format, zur Darstellung von strukturierten Daten und entstand aus der älteren Standard Generalized Markup Language (SGML). Die Standardisierung erfolgt durch das World-Wide-Web-Konsortium (W3C).[15] Weltweit gilt die Sprache als eine der am häufigsten verwendeten für den Austausch von strukturierten Daten zwischen Computern und zwischen Menschen und Computern.[16]

[14] Vgl. Noorul/Ahmed/Shirazi/Yusop (2015), S. 26.

[15] Vgl. Kersken (2004), Online-Quelle [16.07.2019]

[16] Vgl. Skulschus/Wiederstein/Winterstone (2011), S. 10 f.

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

Abbildung 14: Codebeispiel XML, Quelle: Eigene Darstellung

XML weist im Vergleich zu anderen Auszeichnungssprachen viele Vorteile wie Redundanz, die Möglichkeit zur Selbstbeschreibung und das „XML-Versprechen“ auf. Dieses besagt, dass jedes XML-Dokument mit jedem XML-Tool geöffnet, bearbeitet und geparst werden kann.[17]
Die Syntax und der Aufbau der Sprache kann dem Beispiel aus Abbildung 14 entnommen werden.

Kriterium	Relevanz (1-10)	Gewichtung	Beurteilung (1-10)	Beurteilung gewichtet
Lesbarkeit	10	43 %	8	3,44
Zusammenspiel mit C#	8	35 %	10	3,5
Overhead	5	22 %	3	0,66
Summe	**23**	**100 %**	**21**	**7,6**

Tabelle 2: Evaluierung Extensible Markup Language, Quelle: Eigene Darstellung

3.1.2 JavaScript Object Notation

Wie XML ist JSON ein Datenaustauschformat, das komplett unabhängig von der Programmiersprache ist. Es ist einfach von Menschen zu lesen, zu generieren und basiert auf der Programmiersprache JavaScript.[18]

Die Vorteile von JSON werden am Codebeispiel in Abbildung 15 ersichtlich. JavaScript Object Notation verwendet keine End-Tags, ist schnell und einfach zu lesen und unterstützt zusätzlich Arrays. Neben diesen Vorteilen ist weiter zu erwähnen, dass JSON leicht von C# mittels eigenem Framework für .Net geparst werden kann.[19]

[17] Vgl. Quin (2018), Online-Quelle [16.07.2018]

[18] Vgl. json (2018), Online-Quelle [16.07.2018]

[19] Vgl. w3school (2018), Online-Quelle [16.07.2018]

```
{"employees":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
]}
```

Abbildung 15: Codebeispiel JavaScript Object Notation, Quelle: Eigene Darstellung

Kriterium	Relevanz (1-10)	Gewichtung	Beurteilung (1-10)	Beurteilung gewichtet
Lesbarkeit	10	43 %	9	3,87
Zusammenspiel mit C#	8	35 %	9	3,15
Overhead	5	22 %	10	2,2
Summe	**23**	**100 %**	**21**	**9,22**

Tabelle 3: Evaluierung JavaScript Object Notation, Quelle: Eigene Darstellung

3.1.3 YAML Ain't Markup Language

YAML ist keine Auszeichnungssprache, strebt aber dennoch danach, besonders gut und einfach lesbar zu sein. Der Fokus bei dieser Sprache liegt auf der Serialisierung und Deserialisierung von Daten.[20]

Die Vorteile von YAML sind ähnlich wie bei JSON die einfache Lesbarkeit, die Portierbarkeit von Daten zwischen verschiedenen Programmiersprachen und die einfache Implementierbarkeit in bestehende Software. Zusätzlich ist zu erwähnen, dass jeder JSON-Code ein gültiger YAML-Code ist, nicht aber jeder YAML-Code den Standards von JSON entspricht.[21]

```
---
employees:
- firstName: John
  lastName: Doe

- firstName: Anna
  lastName: Smith

- firstName: Peter
  lastName: Jones
```

Abbildung 16: Codebeispiel YAML Ain't no Markup Language, Quelle: Eigene Darstellung

[20] Vgl. YAML (2018), Online-Quelle [16.07.2018]

[21] Vgl. YAML (2018), Online-Quelle [16.07.2018]

Kriterium	Relevanz (1-10)	Gewichtung	Beurteilung (1-10)	Beurteilung gewichtet
Lesbarkeit	10	43 %	10	4,3
Zusammenspiel mit C#	8	35 %	7	2,45
Overhead	5	22 %	10	2,2
Summe	**23**	**100 %**	**21**	**8,95**

Tabelle 4: Evaluierung YAML Ain‘t no Markup Language, Quelle: Eigene Darstellung

3.1.4 Evaluierungsergebnis

Das Zusammenspiel von XML, einer der weltweit meistgenützten Auszeichnungssprachen, mit der Programmiersprache C# ist besonders gut; XML gilt aber im Vergleich mit YAML und JSON als weniger gut lesbar und hat einen vergleichsweise großen Overhead. YAML stellt sich in der Analyse mit der besten Lesbarkeit der drei Sprachen, einem minimalen Overhead, aber keinem eigenen Framework für C# als genaues Gegenstück heraus. Laut dem Ergebnis der Nutzwertanalyse verbindet JSON daher die drei Kriterien Lesbarkeit, Zusammenspiel mit der Programmiersprache C# und einen geringen Overhead am besten und wird daher als Sprache verwendet.

3.2 Automatische Generierung der Parameterdatei

Trotz der guten Lesbarkeit von JSON ist das Erstellen einer Parameterdatei zeitaufwändig und fehleranfällig. Eine Möglichkeit, diese automatisch zu generieren, bietet das von Jungheinrich Systemlösungen verwendete Entwicklertool. Dieses wurde von Entwicklern der Steuerungs-Abteilung programmiert und hilft bei der automatischen Erstellung des Programmes für die speicherprogrammierbaren Steuerungen. Zu Beginn jedes Projektes müssen dafür alle Informationen der Anlage über eine graphische Benutzeroberfläche in das Tool eingegeben werden. Nach dieser manuellen Eingabe sind alle Daten über die Anlage, die vom Emulator benötigt werden – wie etwa Meldepunkte, mögliche Ziele, Anzahl an Lagerfahrzeugen und das Lagerlayout – gespeichert. Das Exportieren einer JSON-Datei aus dem Tool ist dadurch einfach möglich.

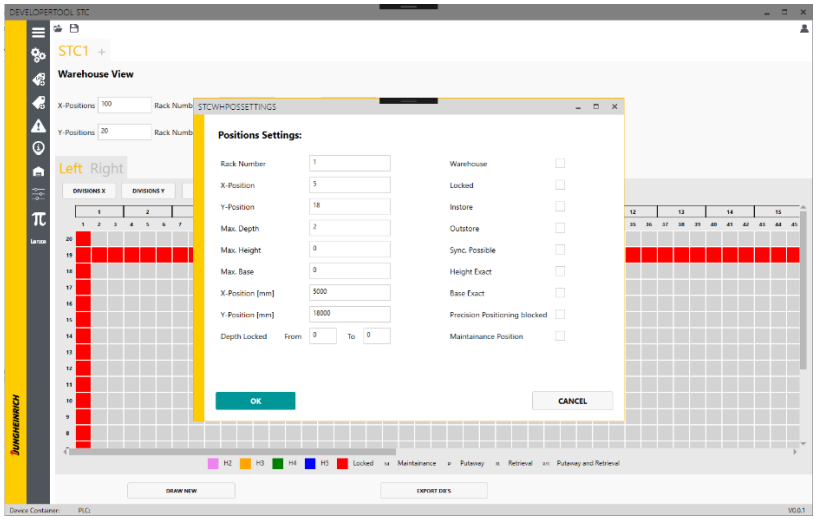


Abbildung 17: Entwicklertool der Steuerungstechnik bei Jungheinrich, Quelle: Eigene Darstellung

4 SOFTWAREARCHITEKTUR-PLANUNG

Inhalt des aktuellen Kapitels ist es, die Softwarearchitektur des WMS-Emulators zu definieren. Dabei werden zunächst verschiede Vorgehensmodelle der Softwareentwicklung beschrieben und das für die Entwicklung des Emulators am besten geeignete ausgewählt. Im Anschluss werden die in der Arbeit verwendeten Architekturmuster erklärt und das Entwicklungsprojekt aufgrund seiner Größe in einzelne Teilsysteme gegliedert und deren Schnittstellen zueinander definiert. Weiters behandelt das Kapitel die für die Entwicklung des Emulators verwendeten Modultests.

4.1 Vorgehensmodell der Softwareentwicklung

Um Aufgaben lösen zu können, ohne auf etwaige Teilschritte zu vergessen, werden standardisierte Verläufe, sogenannte Vorgehensmodelle, verwendet. Diese Modelle zielen darauf ab, Gesamtaufgaben zu gliedern und basieren in der Praxis auf Phasenkonzepten. Diese Phasen bestehen aus einer Vielzahl an Aktivitäten und werden meist sequentiell oder iterativ abgearbeitet. Die Tabelle 5 stellt eine Übersicht der Vorgehensmodelle dar, wobei sich die Modelle in ihrer Ablaufgestaltung und Formalisierung unterscheiden.[22] [23]

		Ablaufgestaltung	
		Sequentiell	**Iterativ**
Formalisierung	**Stark formalisiert**	• Wasserfallmodell • V-Modell • V-Modell XT • W-Modell	• Spiralmodell • Prototyping • OO Lifecycle-Modell
	Wenig formal		• Extreme Programming • SCRUM

Tabelle 5: Vorgehensmodelle zur Softwareentwicklung, Quelle: Eigene Darstellung

Jungheinrich Systemlösungen verwendet für die Entwicklung von Softwarelösungen das V-Modell, wobei dieses in der Praxis meist durch agile Methoden leicht abgewandelt wird. Aus diesem Grund wird auch häufig vom agilen V-Modell gesprochen. Die folgenden Unterkapitel gehen auf das allgemeine V-Modell, agile Ansätze und die Zusammenführung dieser kurz ein.

[22] Vgl. Sandhaus/Berg/Knott (2014), S. 26.

[23] Vgl. Wieczorrek/Mertens (2008), S. 53 f.

4.2 V-Modell

Das V-Modell ist ein Vorgehensmodell, das die einzelnen Phasen des Softwareentwicklungsprozesses definiert. Es wird in spezifische Submodelle mit den Aufgaben Systemerstellung, Qualitätssicherung sowie Konfigurations- und Projektmanagement unterteilt. Auf der absteigenden Seite des V-Modells (links) sind alle mit der Softwareerstellung befassten Aufgaben angeordnet. Diese reichen von ganz allgemeinen Anforderungen bis zu sehr detaillierten Implementierungsgrundladen. Die aufsteigende Seite (rechts) behandelt Implementierungs- und Qualitätssicherungsaufgaben, in denen die Spezifikationen der linken Seite überprüft und getestet werden. Die Abbildung 18 zeigt den Aufbau des V-Modells - hier wird auch die V-Form der Darstellung sichtbar.[24]

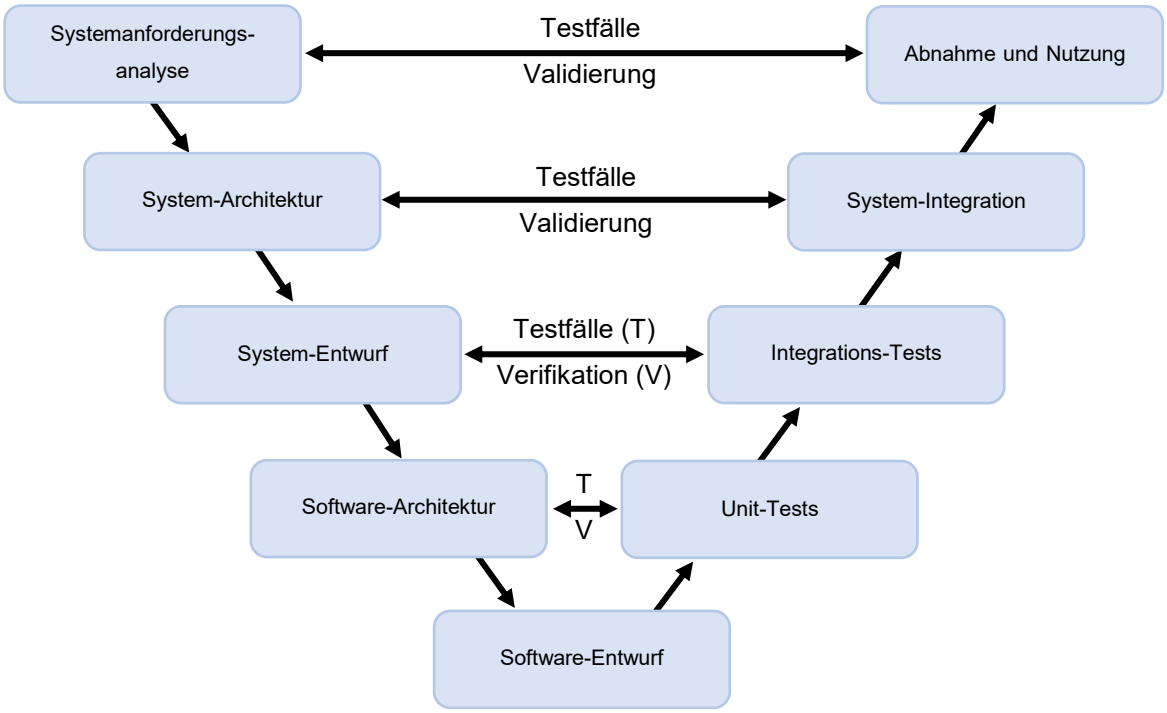


Abbildung 18: V-Modell, Quelle: Eigene Darstellung

Das V-Modell wurde seit seiner Entwicklung 1992 aufgrund neuer Erkenntnisse schon mehrfach überarbeitet. Seit 2004 wird ist es unter dem Namen V-Modell Extreme Tailoring (XT) bekannt und ersetzt damit das V-Modell 97.[25] Dieses Modell unterscheidet sich vom klassischen durch die vermehrte Orientierung an agilen und inkrementellen Ansätzen und die Anpassungsfähigkeit an die jeweiligen Bedürfnisse, wodurch es auch für kleinere Projekte einsetzbar wird.[26]

[24] Vgl. Sandhaus/Berg/Knott (2014), S. 32.

[25] Vgl. Klaus (2018), Online-Quelle [26.07.2018]

[26] Vgl. Sandhaus/Berg/Knott (2014), S. 32.

4.3 Agile Softwareentwicklung

Agile Vorgehensweisen in der Softwareentwicklung erhöhen die Flexibilität während des Entwicklungsprozesses und sollen zu schnelleren Ergebnissen führen. Das Risiko, das durch ungenaue oder sich ändernde Anforderungen entsteht, wird dadurch möglichst gering gehalten.

Um eine gemeinsame Basis sowie einen Leitfaden für die agile Softwareentwicklung zu entwickeln, trafen sich 2001 Begründer verschiedenster agiler Vorgehensmodelle. Dabei entstand das sogenannte „Agile Manifesto“, das vier Werte und zwölf Prinzipien der agilen Softwareentwicklung definiert. Es gilt als Reaktion auf die teils bürokratischen und formalen Prozesse der sequentiellen Vorgehensmodelle.[27]

Agile Werte: [28]

1. „Individuen und Interaktionen wertvoller als Prozesse und Werkzeuge“
2. „Funktionierende Software wertvoller als umfassende Dokumentation“
3. „Zusammenarbeit mit dem Kunden wertvoller als Vertragsverhandlung“
4. „Reagieren auf Veränderung wertvoller als das Befolgen eines Plans“

Die wichtigsten Vorgehensmodelle, die auf diesen Werten basieren, sind Extreme Programming (XP) und SCRUM.

Extreme Programming:
Extreme Programming beschäftigt sich speziell damit, dass Software oft an sich verändernde Anforderungen oder Designänderungen angepasst werden muss und sich somit in einem ständig verändernden iterativen Prozess befindet. Das Lösen der Programmieraufgabe steht bei diesem Modell immer an oberster Stelle, wobei sich die Software in kleinen Schritten den Anforderungen annähert. Das Gesamtprojekt wird dafür in kleine Teilschritte unterteilt. Der Kunde darf in jedem Teilschritt neue Anforderungen an das System nennen. Die Entwickler fügen die geforderten Funktionen hinzu, testen sie und erweitern gleichzeitig den Entwurf des Systems, um alle Funktionen zu unterstützen.[29]

SCRUM:
SCRUM wird von den Begründern als Framework bezeichnet, das speziell für Projekte geschaffen ist, in denen schnell eine Lösung gefunden werden muss oder in denen die Anforderungen noch nicht klar definiert sind. Es basiert auf dem Gedanken, dass die meisten Projekte zu komplex und umfangreich sind, um sie vollständig planen zu können. Ähnlich wie bei XP sind die Anforderungen an das System am Projektstart noch nicht vollständig definiert und es werden auch hier Zwischenergebnisse geschaffen.[30]

---

[27] Vgl. Sandhaus/Berg/Knott (2014), S. 33 ff.

[28] Beck/Beedle/van Bennekum/Cockburn/Cunningham/Fowler/Grenning/Highsmith/Hunt/Jeffries/Kern/Marick/Martin/Mellor /Schwaber/Sutherland/Thomas/ Schwaber (2001), Online-Quelle [01.08.2018]

[29] Vgl. Sandhaus/Berg/Knott (2014), S. 38 f.

[30] Vgl. Sandhaus/Berg/Knott (2014), S. 39 f.

Anders als beim Extreme Programming wird dabei nicht nur das Produkt im iterativen Prozess verfeinert, sondern auch der Plan im Hintergrund, der sogenannte Product Backlog.[31] [32]

4.4 Hybrides Vorgehensmodell

Das Ziel von hybriden Vorgehensmodellen ist die Zusammenführung von sequentiellen und iterativen Abläufen, um ein Modell zu erhalten, das die Stärken beider Denkweisen vereint. So wird zum Beispiel das klassische V-Modell mit agilen Methoden, wie sie Extreme Programming verwendet, verknüpft. Ein besonderes Merkmal des V-Modells ist etwa, dass die Anforderungen seitens des Kunden bereits zu Beginn des Projektes erhoben werden müssen und auf etwaige Veränderungen zu einem späteren Zeitpunkt nicht mehr eingegangen werden kann. Entspricht das gewünschte Produkt am Ende des Entwicklungsprojekts nicht den Wünschen des Kunden, so sind Änderungen daran nur mit großem Mehraufwand möglich. Agile Modelle bieten hier den Vorteil, dass sie während der Erstellung wiederholt Kundenfeedback einholen und so schnell und einfach reagieren können. Ungeplante Änderungen und Erweiterungen können dabei aber in Konflikt mit dem vereinbarten Preis und den vertraglich bestimmten Anforderungen an die Software stehen.[33]

Das hybride Modell versucht den Projektverlauf so zu gestalten, dass die Vorteile beider Ansätze genützt werden können. Dafür sollen zu Beginn des Projektes Anforderungen seitens des Kunden erhoben und vertragliche Rahmenbedingungen festgelegt werden. Die Entwicklungsphase folgt anschließend iterativ in Teilschritten. Am Ende jedes Teilschrittes erfolgt ein Review mit dem Kunden, in dem er das Produkt testen und auf verschiedene Anforderungsänderungen hinweisen kann. Dadurch kann auf sich verändernde Wünsche des Kunden schnell eingegangen und Fehlentwicklungen könne vermieden werden. Nach dem letzten Teilschritt erfolgt die Kundenabnahme und mit ihr die Auslieferung des Produktes. [33]

Diese Vorgehensweise ermöglicht die Vereinbarung eines Festpreises durch die frühe Erhebung der Anforderungen. Spätere kleine Veränderungen können aufgrund der iterativen Annäherung an das Endprodukt trotzdem schnell erkannt werden.

Da sich auch während der Entwicklung des WMS Emulators Anforderungen verändern können und dennoch eine Planung im Vorfeld notwendig ist, um eine genaue Stundenabschätzung abgeben zu können, stellt sich ein hybrides Vorgehensmodell als geeignet heraus. Es sollen dabei wie in

Abbildung 19 mehrere Sprints gebildet werden, wobei der erste Sprint eine verlängerte Analyse- und Designphase aufweist. In ihm sollen die allgemeinen Anforderungen an den Emulator definiert und ein Grunddesign erstellt werden. In den folgenden Sprints werden diese sowie das Produkt an sich weiterentwickelt. Am Ende jedes Sprints wird ein Prototyp generiert, der mit der Abteilungsleitung durchbesprochen wird, die als Product Owner des Projekts fungiert. Das Backlog spiegelt die mitwachsenden Anforderungen an das System wider, das während jedes Sprints erweitert wird.

[31] Vgl. Sandhaus/Berg/Knott (2014), S. 39 f.

[32] Vgl. Gloger (2008), S. 20 f.

[33] Vgl. Sandhaus/Berg/Knott (2014), S. 53 ff.

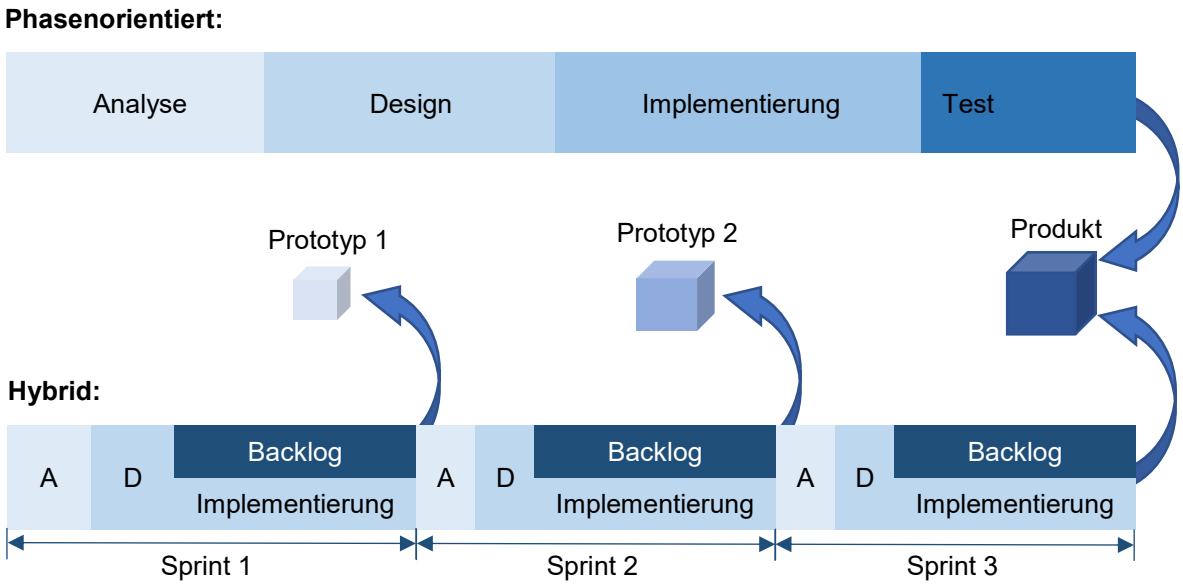


Abbildung 19: Vergleich phasenorientiertes Vorgehensmodell und hybrides Vorgehensmodell, Quelle: Eigene Darstellung

4.5 Architekturmuster

Für das Verständnis der weiterführenden praktischen Kapitel ist es notwendig, gewisse Muster und Techniken der Softwareentwicklung zu kennen. Als Erstes wird das Model View ViewModel (MVVM) Entwurfsmuster, das eine Trennung zwischen graphischer Oberfläche und Logik ermöglicht, näher beschrieben. Anschließend sollen die Vorteile der Objektrelationalen Abbildung für die Programmierung erarbeitet werden.

4.5.1 Model View ViewModel (MVVM)

Das MVVM ist ein Architekturmuster für graphische Benutzeroberflächen und entstand 2005 mit der Entwicklung von Windows Presentation Foundation (WPF). Es dient der Trennung von Darstellung und Logik, wodurch die graphische Oberfläche der Anwendung getrennt von der Logik entwickelt werden kann.[34]

Die Trennung des Entwurfsmusters geschieht wie in Abbildung 20 ersichtlich in drei Schritten: der View, dem ViewModel und dem Model. Das Model beinhaltet die Daten, die vom Benutzer verändert und abgerufen werden können. Es ist verantwortlich für die Datenerhaltung und die Geschäftslogik. In der View wird die Oberfläche des Systems in der Sprache XAML modelliert. Die View ist rein visuell, beinhaltet somit keinerlei Logik und erlaubt dem Benutzer, den Status des Programmes zu erkennen und zu verändern. Das ViewModel trennt die View vom Model; es beinhaltet Datenfelder, die mittels Databinding an die Kontrollelemente der View gebunden, sowie Ereignisse, die von Kontrollelementen der View ausgelöst werden können. Wird ein Datenfeld des ViewModels verändert, so erhält das Kontrollelement der View eine Benachrichtigung über deren Änderung und aktualisiert diese automatisch. Die Verbindung zwischen der

[34] Vgl. McLean Hall (2010), S. 55

View und dem ViewModel geschieht über die „DataContext“-Eigenschaft, die die Bindung aller Kontrollelemente der View an die Datenfelder des ViewModels ermöglicht.[35]

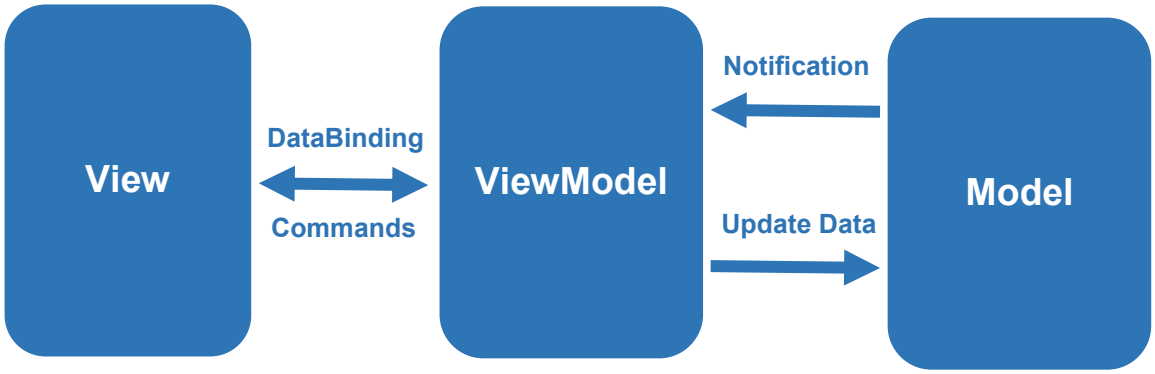


Abbildung 20: MVVM Konzept, Quelle: Eigene Darstellung

Der Vorteil des MVVM Architekturmusters liegt in der Austauschbarkeit der View, ohne Änderungen am ViewModel oder am Model vorzunehmen. Durch das Konzept des Databindings befindet sich kein Code in der View, wodurch diese von UI-Designern erstellt werden kann, während Entwickler am ViewModel und am Model arbeiten können. Als weiterer Vorteil gegenüber anderen Architekturmustern gilt die Testbarkeit der Oberfläche. Unittests können direkt für das ViewModel entworfen und müssen nicht händisch über die graphische Benutzeroberfläche erledigt werden.

Als Nachteil stellt sich der zu Beginn erhebliche Entwicklungsaufwand, der für das MVVM notwendig ist, heraus. Dieser Aufwand rechnet sich erst bei der Entwicklung größerer Anwendungen und ist für kleinere Projekte nicht zu empfehlen.

4.5.2 Object Relational Mapping (ORM) und das Entity Framework

ORM ermöglicht Entwicklern, die mit objektorientierten Programmiersprachen arbeiten, Daten in einer relationalen Datenbank abzulegen. Die Tabellen und Spalten der Datenbank können dabei wie Objekte und Eigenschaften angesprochen werden. Die relationale Datenbank verhält sich wie eine objektorientierte Datenbank und erleichtert somit die Entwicklung.

Das Entity Framework von Microsoft kann als objektrelationaler Mapper eingesetzt werden und nimmt dem Entwickler so den Großteil an Datenbankzugriffcodes ab. Voraussetzung dafür ist der Einsatz von .Net Programmiersprachen wie C# oder VB. Der Datenbankzugriff erfolgt über ein Modell, das aus Entitätsklassen und einem abgeleiteten Kontext besteht. Dieses Modell kann auf drei verschiedene Weisen erstellt werden:[36]

- Database-first design: Das Modell wird aus einer bereits vorhandenen Datenbank erstellt.[37]

[35] Vgl. Smith (2010), S. 9 ff.

[36] Vgl. Microsoft (2016), Online-Quelle [02.09.2018]

[37] Vgl. Lerman (2010), S. 11.

- Model-first design: Das Modell wird unter Zuhilfenahme des Entity-Model-Designers entwickelt. Die Datenbank wird anschließend direkt aus dem Modell erstellt.[37]
- Code-first design: Die Klassen der Datenbank werden händisch entwickelt, daraus lässt sich anschließend die Datenbank erstellen.[37]

Das Abfragen von Daten aus der Datenbank ist mittels Language Integrated Query (LINQ) möglich, die nach der Übergabe an die Datenbank in ein SQL Statement übersetzt wird. Ein Beispiel für solch eine Abfrage kann der Abbildung 21 entnommen werden. Darin wird ein Tabelleneintrag der Tabelle „Pallet“ aus der Datenbank geladen, dessen Identifikationsnummer gleich eins sein muss.[38]

```
using (var context = new WarehouseContext())
{
    var pallets = context.Pallets
        .Single(p => p.id == 1);
}
```

Abbildung 21: Abfragen von Daten aus einer Datenbank mit dem Entity Framework, Quelle: Eigene Darstellung

Alle Änderungen an den Instanzen der Entitätsklassen werden innerhalb eines ChangeTracker Objektes gespeichert, um nach Aufrufen der SafeChanges Methode in die Datenbank geschrieben zu werden. Die SafeChanges Methode wird dann vom Datenbankanbieter zum Beispiel in die Befehle Insert, Update und Delete übersetzt. Die Abbildung 22 zeigt ein Beispiel für das Hinzufügen eines Datensatzes zur Entitätsklasse und das anschließende Speichern in der Tabelle „Pallet“.[38]

```
using (var context = new WarehouseContext())
{
    var pallet = new Pallet { id = "AB1C3432F" };
    context.Pallets.Add(pallet);
    context.SaveChanges();
}
```

Abbildung 22: Speichern von Daten in einer Datenbank mit dem Entity Framework, Quelle: Eigene Darstellung

4.6 Systemteil Definition

Der Emulator soll aufgrund seiner Größe und Komplexität in die drei folgenden Pakete aufgeteilt werden: die Datenbank, die Anwendung und die grafische Benutzeroberfläche. Jedem dieser drei Systemteile werden eigene Aufgaben und Anforderungen zugeordnet. Inhalt dieses Kapitels ist es, diese zuzuteilen und die Schnittstellen zwischen den Paketen zu definierten. Die Definition dieser Aufgabengebiete und Schnittstellen soll bei der getrennten Entwicklung der Systemteile helfen. Die genauen Anforderungen und Funktionen der Pakete werden in den Kapiteln 5, 6 und 7 bearbeitet.

[38] Vgl. Microsoft (2016), Online-Quelle [02.09.2018]

4.6.1 Datenbank

Die Hauptaufgaben der Datenbank sind das strukturierte Ablegen sowie die Sicherung der Daten des Emulators. Zu diesen Daten zählen Anlagenparameter wie die Anzahl und Anordnung von Meldepunkten und Anlagendaten wie die aktuelle Position von LHMs, der Status von Verbindungen und freie Lagerplätze.

Bei der Entwicklung dieses Paketes sind die Tabellen der Datenbank zu entwerfen, die Schlüssel der Tabellen zu definieren und die Verbindungen zwischen den Tabellen anzulegen. Neben der Datenbank selbst zählt auch der Zugriff auf die gespeicherten Daten zu diesem Systemteil.

4.6.2 Anwendung

Die Anwendung beinhaltet die Kernfunktionen des Emulators. Dieses Paket ist für die Logik des Systems zuständig und imitiert die Funktionen des nachzuahmenden Warehouse-Management-Systems. Die Erstellung dieses Teils erfolgt in der Programmiersprache C#, da mit Hilfe dieser Sprache eine schnelle Entwicklung erfolgen kann und C# von Jungheinrich Systemlösungen als Standard-Programmiersprache in der Steuerungstechnikabteilung verwendet wird.

Die Aufgaben dieses Systemteils sind die Kommunikation mit der Steuerungsebene sowie alle Funktionen des zu emulierenden WMS, die für das Testen der Steuerungsebene notwendig sind.

4.6.3 Grafische Benutzeroberfläche

Die Oberfläche ermöglicht dem Benutzer des Emulators vorhandene Funktionen auszulösen und Informationen über dessen aktuellen Zustand zu erhalten. Die Entwicklung dieses Teils erfolgt mit Windows Presentation Foundation (WPF), da dieses Framework die Trennung von Design und Oberflächenlogik ermöglicht, und die Logik, wie schon die Anwendung des Emulators selbst, in C# programmiert werden kann. Das Design der Oberfläche soll an das Jungheinrich Warehouse-Management-System angepasst werden und eine einfache, benutzerfreundliche Bedienung erlauben.

4.6.4 Schnittstellendefinition der Systemteile

Für die getrennte Entwicklung der Pakete müssen im Vorfeld deren Schnittstellen definiert werden. Als Schnittstelle dient das Entity Framework von Microsoft, das den Zugriff von objektorientierten Anwendungen auf Daten von relationalen Datenbanken erleichtert. Sowohl die Anwendung als auch die grafische Benutzeroberfläche kommunizieren mit der Datenbank über dieses Framework. Eine direkte Schnittstelle zwischen den beiden soll es nicht geben, damit die Oberfläche auf mehreren Rechnern gestartet werden kann und so die gleichzeitige Bedienung der Anlage für verschiedene Inbetriebnehmer möglich wird. Der Datenaustausch beider Systeme erfolgt dabei ausschließlich über die Datenbank. Der Abbildung 23 kann der schematische Aufbau der Softwarepakete entnommen werden.

Um mit der Entwicklung der grafischen Oberfläche und der Anwendung beginnen zu können, wird ein grober Aufbau der Datenbank benötigt. Aus diesem Grund wird mit der Konzeption und Erstellung der Datenbank begonnen.

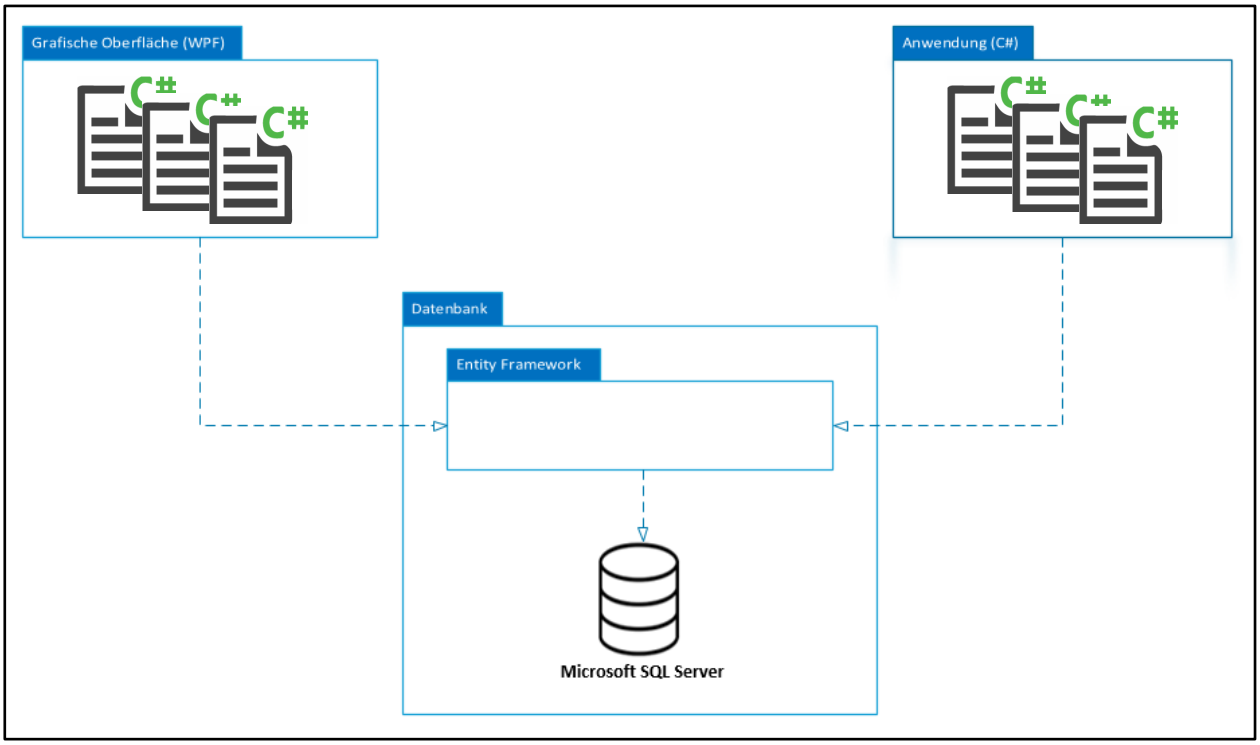


Abbildung 23: Systemteile des Emulators, Quelle: Eigene Dartstellung

5 DATENBANK

In der Datenbank können alle Parameter und Daten des WMS-Emulators strukturiert abgelegt werden. Sie dient so als Schnittstelle zwischen der grafischen Benutzeroberfläche und der Anwendung. Im folgenden Kapitel werden zuerst die Anforderungen an die Datenbank definiert. Anschließend werden davon der Aufbau der Tabellen sowie die Beziehungen zwischen diesen abgeleitet. Nach dem Design der Datenbank wird abschließend der Einsatz des Microsoft Entity Frameworks im Emulator behandelt.

Im Rahmen dieser Arbeit wird der Microsoft SQL Server verwendet, da dieser von Jungheinrich Systemlösungen bereits eingesetzt wird und somit genügend Lizenzen und Knowhow im Unternehmen vorhanden sind.

5.1 Anforderungen

Um mit der Entwicklung der Datenbank beginnen zu können, müssen zunächst die Anforderungen an sie definiert werden. Diese wurden in Abstimmung mit den Entscheidungsträgern von Jungheinrich Systemlösungen erarbeitet. Folgende Anforderungen sollen demnach bei der Erstellung der Datenbank eingehalten werden:

- Übersichtliche Datenstruktur
- Die Redundanz der Daten soll so gering wie möglich gehalten werden, um Anomalien in der Datenbank zu vermeiden
- Objektorientierter Zugriff mittels MS Entity Framework
- Es sollen keine Prozeduren in der Datenbank programmiert werden. Für eine bessere Übersichtlichkeit sollen alle Funktionen in der Anwendung des Emulators realisiert werden
- Die Konfiguration der Anwendungssoftware soll vollständig aus der Datenbank geladen werden können

Neben den allgemeinen Anforderungen an das Design der Datenbank sollen die Anforderungen an die zu speichernden Daten definiert werden. Diese können in folgende Gruppen unterteilt werden und enthalten Konfigurationsdaten und aktuelle Daten wie Status oder Positionen von Ladehilfsmitteln:

Ladehilfsmittel:

Die Datenbank soll Informationen über die Ladehilfsmittel, wie ihre aktuelle Position, ihre Größe, ihr Gewicht und ihren Barcode, speichern.

Meldepunkt:

Der Name des Meldepunktes, sein Status, Typ und einschränkende Eigenschaften, wie die maximale Auslastung des Punktes, werden in der Datenbank abgelegt. Zusätzlich soll der Punkt einer Steuerung sowie einem Ort zuordenbar sein.

Verbindungen:

Informationen, die für den Verbindungsaufbau des Emulators zur Steuerungsebene benötigt werden, sollen in der Datenbank gesichert werden können.

Routen:

Alle möglichen Routen der Anlage werden inklusive ihrer Anfangs- und Endpunkte gespeichert. Auch weitere Informationen, wie der aktuelle Status, die Länge der Route und einschränkende Eigenschaften für die Wahl der Route, sollen in der Datenbank abgelegt werden können.

Aufträge:

Alle vom Benutzer erstellten Aufträge können in der Datenbank für die Weiterverarbeitung gesichert werden. Die Daten des Auftrages sollen sich aus dem aktuellen Status des Auftrages, dem Anfangs- und Endzeitpunkt sowie der Zuordnung zu den Ladehilfsmitteln zusammensetzen.

Fehlermeldungen:

Alle aktuellen Fehler der Anwendung und Steuerungsebene mit Zeitpunkt des Auftretens, dem betroffenen Meldepunkt und dem Fehlertext sollen in der Datenbank abgelegt werden können.

5.2 Aufbau der Tabellen

Im folgenden Kapitel werden die Anforderungen an die zu erstellenden Tabellen einzeln definiert und deren Aufbau bestimmt.

Tabelle Ladehilfsmittel (LHM)

Jedes Ladehilfsmittel muss über eine eindeutige Nummer identifizierbar sein. Der Barcode gilt aufgrund seiner Fehleranfälligkeit nicht als eindeutig, sondern nur als Attribut des LHM. Neben den Attributen zur Identifikation sollen auch Informationen wie Größe, aktuelle Position und Zielposition in der Tabelle enthalten sein. Die aktuelle Position ist über den aktuellen Meldepunkt, auf dem sich das LHM befindet, den aktuellen Wegabschnitt oder ein weiteres LHM, auf dem das LHM gestapelt wurde, bestimmbar. Die Zielposition soll über das nächste Teilziel und das Endziel definiert werden. Konnte das LHM nicht korrekt identifiziert werden, sodass es zum Nicht in Ordnung (NiO) Meldepunkt geschickt werden muss, wird eine Routenbedingung an das LHM angehängt, dass den Meldepunkt bei der Zielsuche unterstützt.

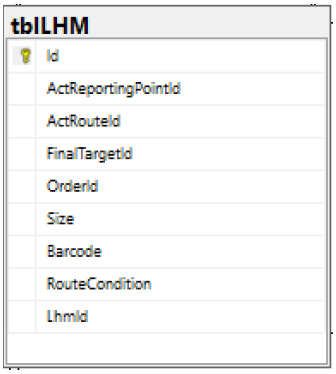


Abbildung 24: LHM Tabelle, Quelle: Eigene Dartstellung

Tabelle Programmable logic controller (Plc)

Jede für das WMS relevante Speicherprogrammierbare Steuerung (SPS) soll in der Datenbank angegeben werden, um die Zuordnung der Meldepunkte und Verbindungen zu erlauben. Ähnlich wie ein Meldepunkt soll auch die Steuerung einen bestimmten Typ annehmen, an den bestimmte Aufgaben geknüpft werden können. Neben diesem Typ soll der aktuelle Status, der Name und die Identifikationsnummer in der Tabelle abgelegt werden.

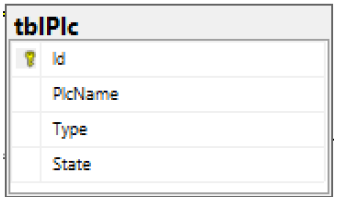


Abbildung 25: Tabelle Plc, Quelle Eigene Darstellung

Tabelle ReportingPoint (Meldepunkt)

Als Meldepunkte werden jene Positionen in der Anlage bezeichnet, an denen ein Telegrammverkehr zur Steuerungsebene stattfindet. Meldepunkte können verschiedene Typen annehmen, wobei jeder Typ unterschiedliche Aufgaben erfüllt. Neben dem Typ des Meldepunktes müssen die Identifikationsnummer, der Name und der aktuelle Status in der Tabelle gespeichert werden. Weiters werden neben der Auslastung der Position auch die maximale Größe sowie der Ort und die Steuerung, der der Meldepunkt zugeordnet ist, gesichert. Ist ein Meldepunkt einem Lagerfach zugeteilt, soll dieses über die Warehouse ID (WhID) mit dem Meldepunkt verknüpft werden können.

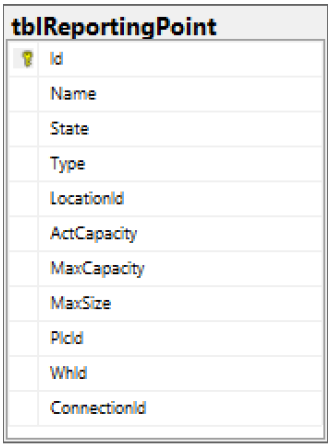


Abbildung 26: Tabelle ReportingPoint, Quelle: Eigene Darstellung

Tabelle ReportingPointLoc (Meldepunkt Ort)

Wie zuvor erwähnt soll jeder Meldepunkt einem genauen Ort zuordenbar sein. In dieser Tabelle befinden sich daher Name und Identifikationsnummer. Die Gruppierung der Meldepunkte soll es ermöglichen, Ladehilfsmittel zu einem bestimmten Ort, wie zum Beispiel Auslagerung, zu schicken.

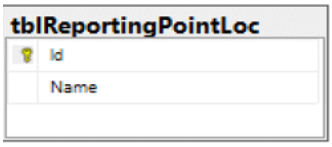


Abbildung 27: Tabelle ReportingPointLoc, Quelle: Eigene Darstellung

Tabelle WhPos (Lagerplatz)

Ein Meldepunkt kann, wie bereits erklärt, einem Lagerfach zugeordnet werden. Diese Tabelle soll diesen genau beschreiben. Über die Spalte „Rack“ wird der Lagerplatz der Lagerzeile, mit „Channel“ der X-Koordinate und mit „Height“ der Y-Koordinate zugeordnet.

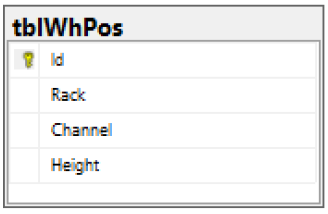


Abbildung 28: Tabelle WhPos, Quelle: Eigene Darstellung

Tabelle Connection (Verbindung)

Jedem Meldepunkt können mehrere Verbindungen zugeordnet werden, über die das WMS mit der Steuerung kommuniziert. Für die eindeutige Beschreibung der Verbindung müssen die IP-Adresse der Steuerung sowie der Port beider Partner angegeben werden. Neben den Adressen ist der Typ der Verbindung für den Verbindungsaufbau notwendig und in der Datenbank abzulegen.

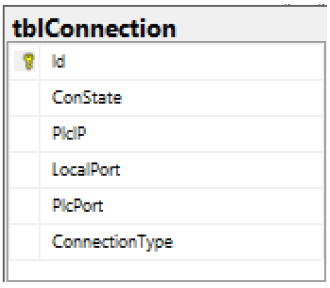


Abbildung 29: Tabelle Connection: Quelle: Eigene Darstellung

Tabelle Route

Die Strecke zwischen zwei Meldepunkten wird als Route bezeichnet. Jede Route hat eine definierte Quelle und ein definiertes Ziel. Die Länge der Strecke soll über einen Distanz-Wert angegeben werden können. Dieser Wert ermöglicht dem Emulator, kurze Wegstrecken langen vorzuziehen. Neben diesen Attributen soll der Status der Strecke sowie deren Bedingung, wenn vorhanden, gespeichert werden. Das Attribut Bedingung erlaubt dem Emulator, LHMs mit derselben Bedingung auf die entsprechende Route zu schicken. Diese Funktion wird speziell für den Weg zu NiO-Plätzen verwendet.

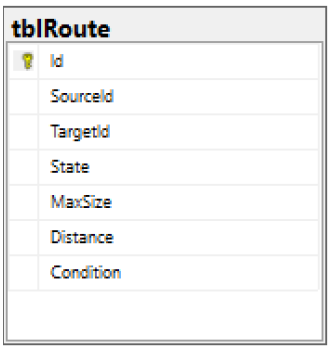


Abbildung 30: Tabelle Route, Quelle: Eigene Darstellung

Tabelle Order

Aufträge sollen vom Benutzer über die grafische Oberfläche angelegt werden können. In der Datenbank werden dafür das Ziel des Auftrages, die Anzahl der auszulagernden sowie der noch nicht ausgelagerten LHMs, der aktuelle Auftragsstatus und Start- und Endzeitpunkt gespeichert.

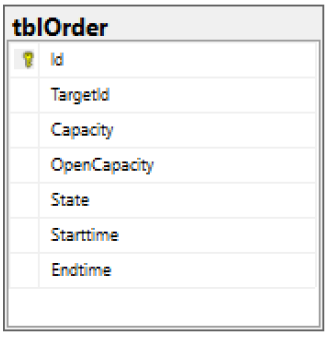


Abbildung 31: Tabelle Order, Quelle: Eigene Darstellung

Tabelle VehicleOrder

Für jedes LHM, das von einem Regalbediengerät bewegt wird, soll ein Fahrzeug-Auftrag erstellt werden. Dabei kann es sich um eine Einlagerung, Auslagerung oder Umlagerung in der Gasse handeln. Ein Fahrzeug-Auftrag soll aus der Quelle, dem Ziel, dem Zeitpunkt des Erstellens, dem Start- und dem Endzeitpunkt bestehen. Die Zuordnung zum RBG erfolgt über die Identifikationsnummer der SPS.

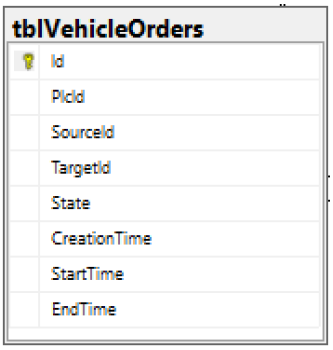


Abbildung 32: Tabelle VehicleOrders, Quelle: Eigene Darstellung

Tabelle ErrorMsg

Die Fehlertexte der Anwendung und der Steuerungsebene sollen in der Datenbank gesichert werden. Eine Fehlermeldung besteht dabei aus der Fehlernummer und dem Fehlertext.

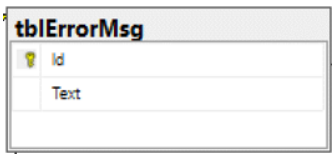


Abbildung 33: Tabelle ErrorMsg, Quelle: Eigene Darstellung

Tabelle Error

Um eine Übersicht über alle Fehler der Anwendung und der Steuerungsebene gewinnen zu können, sollen diese in einer Tabelle abgelegt werden. Die Message ID (MsgId) gibt dabei Auskunft über den Fehlertext und das Attribut TimeStamp über den Fehlerzeitpunkt. Durch die Attribute PlcId und ReportingPointId lässt sich der Fehler dem Meldepunkt oder der Steuerung zuordnen.

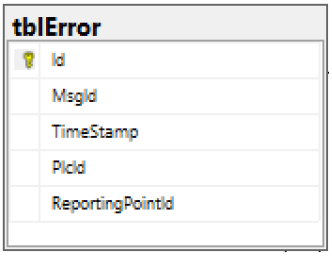


Abbildung 34: Tabelle Error, Quelle: Eigene Darstellung

5.3 Beziehungen zwischen den Tabellen

Wie schon im vorigen Kapitel erwähnt kann es zu Abhängigkeiten zwischen den Tabellen kommen. Diese werden in der Datenbank als Beziehungen bezeichnet. Inhalt dieses Kapitels ist es, die Beziehungen der im Kapitel 5.2 vorgestellten Tabellen zu definieren. Eine Übersicht über alle Beziehungen zwischen den Tabellen kann der Abbildung 35 entnommen werden.

Um Beziehungen zwischen zwei Tabellen erstellen zu können, werden ein Primärschlüssel und ein Fremdschlüssel benötigt. Die Primärschlüssel der Tabellen wurden mit dem Namen „Id“ bezeichnet.

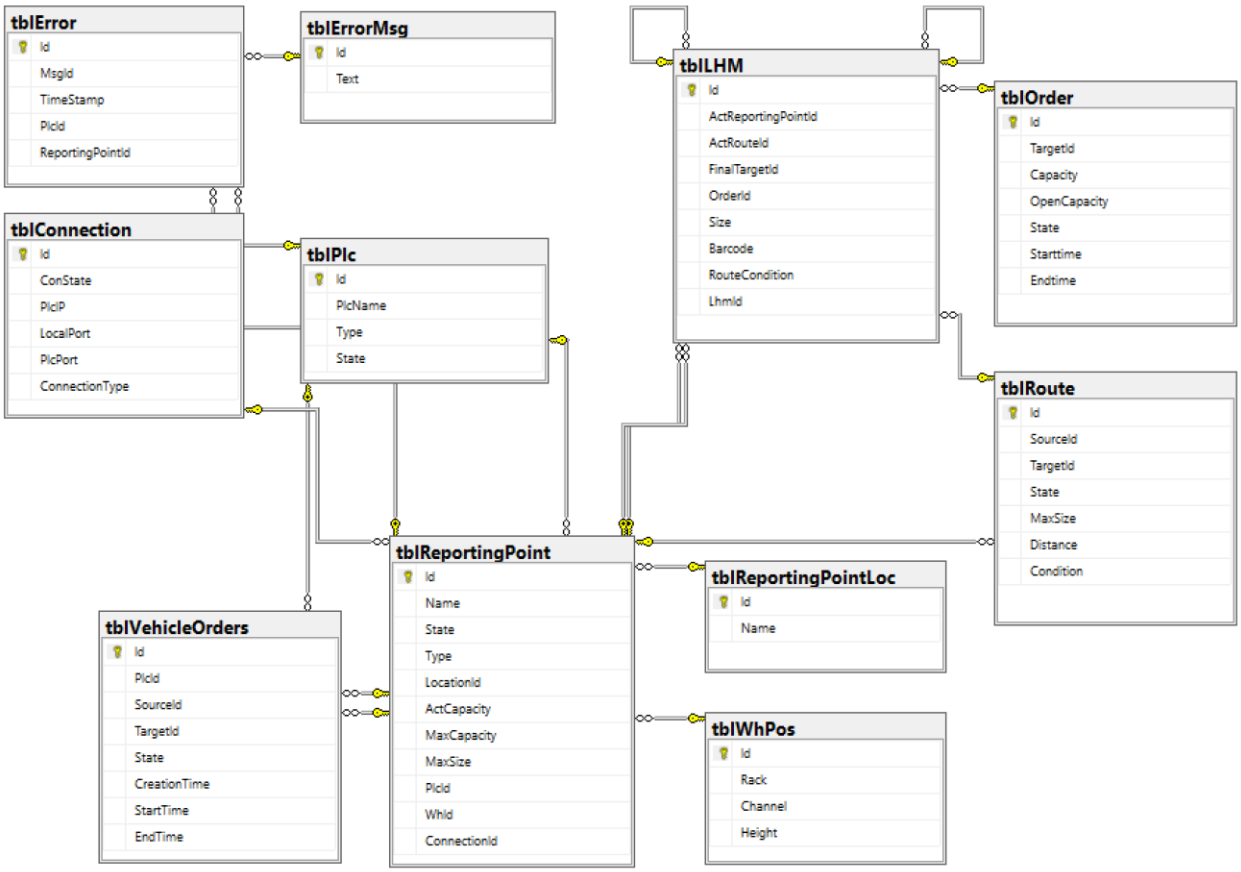


Abbildung 35: Aufbau und Beziehungen aller Tabellen der Datenbank, Quelle: Eigene Darstellung

Tabelle ReportingPoint

Die Tabelle ReportingPoint ist wie in Abbildung 35 ersichtlich die Tabelle mit den meisten Beziehungen. Es wird dabei zwischen 1:n- und n:1-Beziehungen unterschieden. Die Beziehungen zwischen der Tabelle ReportingPoint und den Tabellen VehicleOrder, Route, LHM, Location, WhPos, PLC und ErroMsg werden in Tabelle 6 aufgelistet. Es werden dabei sowohl die Primärschlüssel als auch die Fremdschlüssel aller Beziehungen gezeigt.

1:n-Beziehung		**n:1- Beziehung**	
Primärschlüssel	**Fremdschlüssel**	**Primärschlüssel**	**Fremdschlüssel**
tblReportingPoint: Id	tblVehicleOrder: Source	tblLocation: Id	tblReportingPoint: Location
	tblVehicleOrder: Target	tblWhPos: Id	tblReportingPoint: WhId
	tblRoute: Source	tblPlc: Id	tblReportingPoint: PlcId
	tblRoute: Target	tblErrorMsg: Id	tblReportingPoint: ErrorId
	tblLHM: ActReportingPoint		
	tblLHM: FinalTarget		

Tabelle 6: Beziehungen Tabelle ReportingPoint, Quelle: Eigene Darstellung

Tabelle PLC

Wie bereits in Kapitel 5.2 erwähnt können der Tabelle PLC mehrere Fahraufträge, Meldepunkte und Fehler zugeordnet werden. Der Primärschlüssel der Beziehungen sowie die Fremdschlüssel werden in Tabelle 7 aufgelistet.

1:n-Beziehung		**n:1-Beziehung**	
Primärschlüssel	**Fremdschlüssel**	**Primärschlüssel**	**Fremdschlüssel**
tblPlc: Id	tblVehicleOrder: PlcId		
	tblReportingPoint: PlcId		
	tblErrorMsg: Id		

Tabelle 7: Beziehungen Tabelle Plc: Quelle: Eigene Darstellung

Tabelle Error

Jeder Fehler kann maximal einem Fehlertext sowie einer Steuerung oder einem Meldepunkt zugeordnet werden. Da es daher nur n:1-Beziehungen zwischen der Tabelle Error und den anderen Tabellen gibt, bleibt die linke Seite der Tabelle 8 frei.

1:n-Beziehung		n:1-Beziehung	
Primärschlüssel	**Fremdschlüssel**	**Primärschlüssel**	**Fremdschlüssel**
		tblPlc: Id	tblError: PlcId
		tblReportingPoint: Id	tblError: ReportingPointId
		tblErrorMsg: Id	tblError: MsgId

Tabelle 8: Beziehungen Tabelle ErrorMsg, Quelle: Eigene Darstellung

Tabelle LHM

Die Tabelle LHM hat nur eine 1:n-Beziehungen zu sich selbst und vier n:1-Beziehungen zu anderen Tabellen. Die 1:n-Beziehung zu sich selbst entsteht durch die Möglichkeit, dass sich ein Ladehilfsmittel physisch auf einem anderen Ladehilfsmittel befinden kann, zum Beispiel bei einem LHM-Stapel. Die Schlüssel der Beziehungen können der Tabelle 9 entnommen werden.

1:n-Beziehung		n:1- Beziehung	
Primärschlüssel	**Fremdschlüssel**	**Primärschlüssel**	**Fremdschlüssel**
LHM: Id	LHM: LhmId	LHM: Id	LHM: LhmId
		tblReportingPoint: Id	tblLHM: ActReportingPointId
		tblReportingPoint: Id	tblLHM: FinalTargetId
		tblRoute: Id	tblLHM : ActRouteId
		tblOrder: Id	tblLHM: OrderId

Tabelle 9: Beziehungen Tabelle ReportingPoint, Quelle: Eigene Darstellung

5.4 Microsoft Entity Framework

Die theoretischen Aspekte des Microsoft Entity Frameworks wurden bereits in Kapitel 4.5.2 erläutert. In diesem Kapitel sollen nun die praktischen Gesichtspunkte des Frameworks erarbeitet werden. Wie bereits erwähnt gibt es drei Herangehensweisen, um das EntityModel zu erstellen. Da das Grundmodell der Datenbank bereits mithilfe des Microsoft SQL Server Management Studios entwickelt wurde, soll das Database-first Designmodell verwendet werden.

Dieses Designmodell erstellt automatisch aus einer bereits vorkonfigurierten Datenbank das Entitätsmodell. Daraus werden anschließend mittels Codegenerator Klassen und Eigenschaften generiert. Ihre Namensgebung erfolgt automatisch auf Basis der Namen der Tabellen und Spalten der Datenbank. Einzig die Namen der Beziehungen zwischen den Klassen werden bei der Erstellung neu vergeben. Existieren zwischen zwei Tabellen mehr als eine Beziehung, werden diese lediglich durch eine fortlaufende Nummer unterschieden. Dies kann bei Verwendung der Eigenschaften zu Unklarheiten führen.

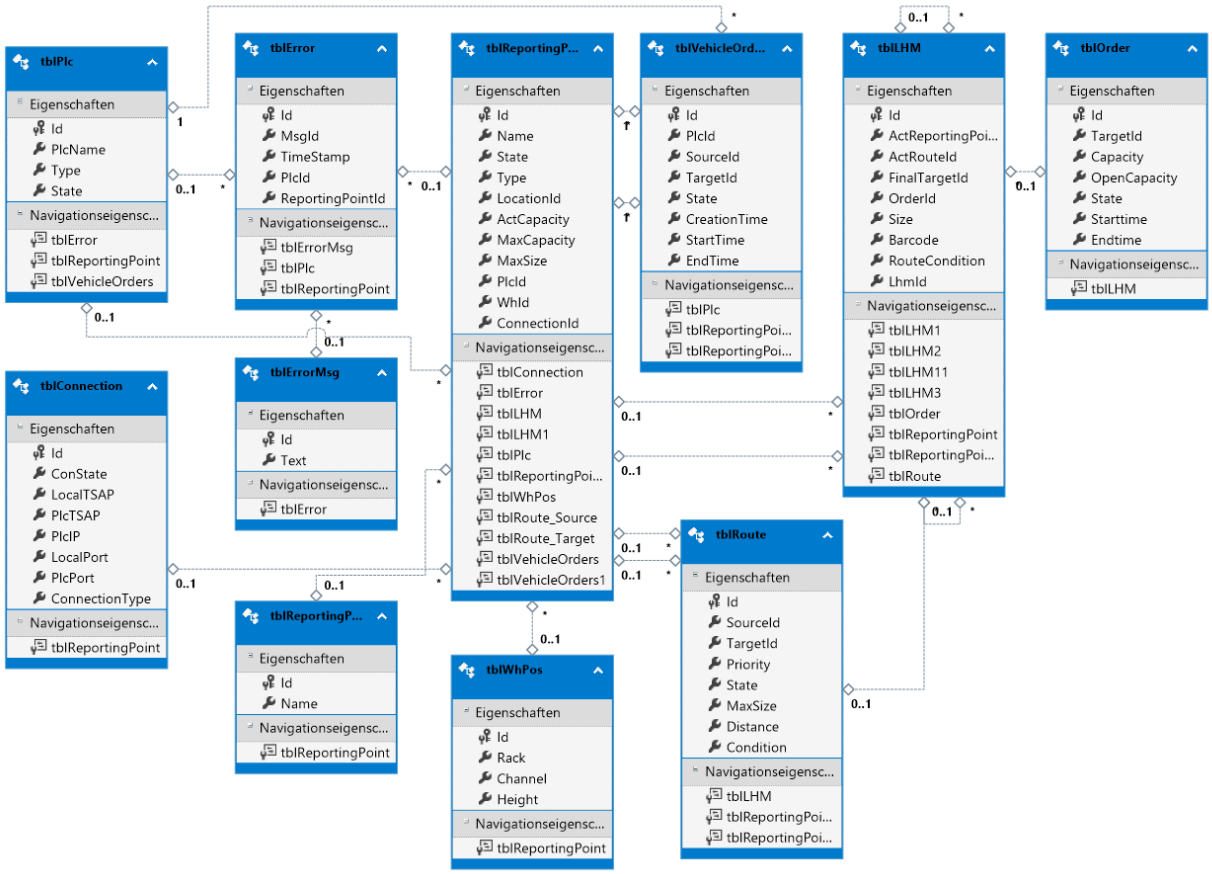


Abbildung 36: Entitätsmodell Visual Studio, Quelle: Eigene Darstellung

5.4.1 Early Prototyping

Um mit dem Erstellen des Anwendungs- und Oberflächenkonzepts beginnen zu können, werden genauere Informationen über die Funktionsweise des Entity Frameworks benötigt. Diese sollen in zwei Versuchen in Erfahrung gebracht werden. Zu untersuchen ist, welche Auswirkungen Veränderungen der Daten in einzelnen Instanzen des Entitätsmodells aufeinander haben.

Versuch 1:

1. Erstellen von zwei Instanzen des Entitätsmodells
2. Ändern der Daten in der ersten Instanz
3. Prüfen, ob die Daten in Instanz zwei automatisch aktualisiert wurden
4. Wurden die Daten nicht automatisch aktualisiert, soll der Versuch wiederholt werden und nach Änderung der Daten in Instanz eins die Methode SafeChanges ausgeführt werden

Abbildung 37 kann entnommen werden, dass die Daten erst nach dem Aufruf der SafeChanges Methode in der zweiten Instanz aktualisiert werden. Auffällig bei diesem Versuch ist, dass nach dem Hinzufügen eines neuen Datensatzes dieser auch in Instanz eins nicht aufscheint, sondern erst nach dem remanenten Speichern in der Datenbank angesprochen werden kann.

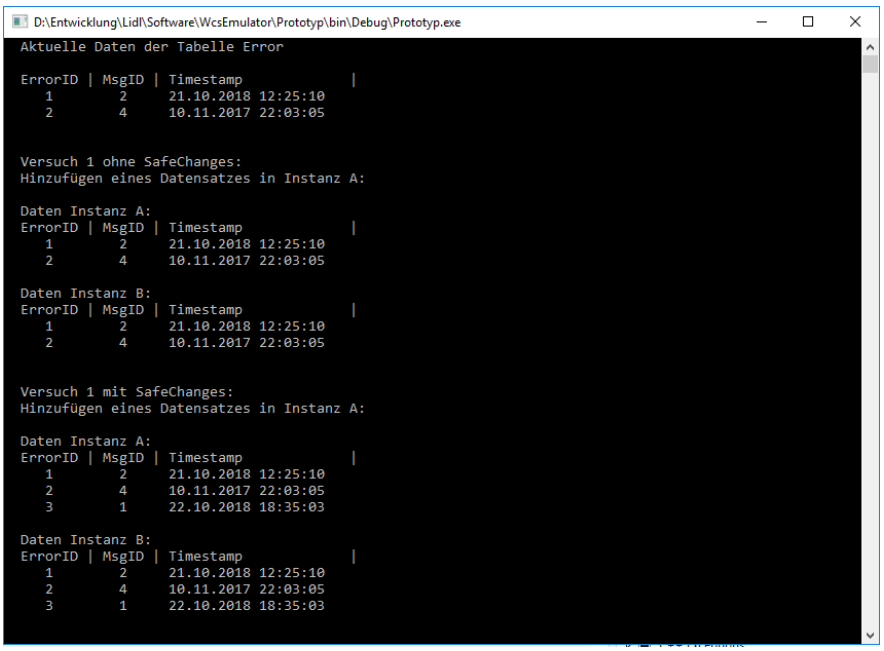


Abbildung 37: Prototyp Entity Framework Versuch 1, Quelle: Eigene Darstellung

Versuch 2:

1. Erstellen von zwei Instanzen des Entitätsmodells
2. Ändern der Daten in der ersten Instanz
3. Ändern der Daten in der zweiten Instanz
4. Speichern beider Instanzen mittels SafeChanges Methode
5. Überprüfen, ob Instanz eins und Instanz zwei korrekt aktualisiert wurde.

Beide Änderungen werden in der Datenbank korrekt übernommen. Weiters konnte festgestellt werden, dass bei Veränderung des gleichen Datensatzes in verschiedenen Instanzen des Entitätsmodells jene gespeichert wird, die als letztes bearbeitet wurde.

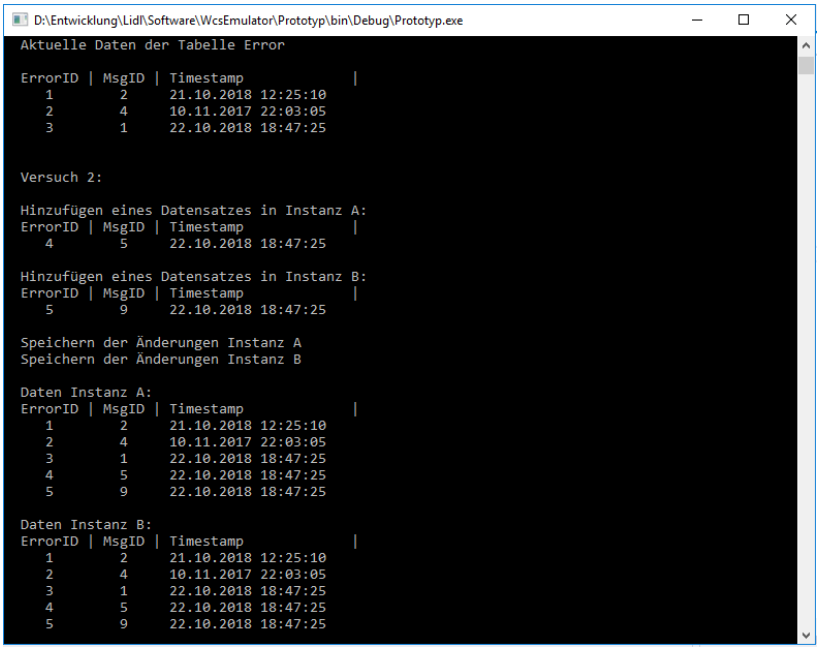


Abbildung 38: Prototyp Entity Framework Versuch 2, Quelle: Eigene Darstellung

5.4.2 Probleme in der Verwendung

Während dem Arbeiten mit dem Framework kann es vor allem zu Beginn zu mehreren Schwierigkeiten kommen. Wie in Kapitel 5.4 bereits erwähnt, werden Beziehungen zwischen zwei Tabellen, sofern mehrere bestehen, lediglich durch eine fortlaufende Nummer unterschieden. Diese unklare Benennung sorgte für Unübersichtlichkeit, die vorerst durch händische vergebene Namen gelöst wurde. Beim Updaten des Entitätsmodells an der Datenbank wurden diese Änderungen allerdings wieder vom Framework überschrieben.

Zu weiteren Problemen kam es beim Updaten des Modells, sobald ganze Tabellen oder Spalten in der Datenbank umbenannt wurden. Die Klassen oder Eigenschaften wurden im Modell nicht angepasst, sondern nach einer Änderung ein weiteres Mal erstellt. Alte Klassen mussten händisch gelöscht werden.

Als passende Lösung konnte der Umstieg auf das Model-first-Designmodell nach der erstmaligen Erstellung des Modells gefunden werden, um Änderungen nicht in der Datenbank vorzunehmen, sondern direkt am Modell. Die Datenbank kann nach Veränderungen durch eine automatische Generierung der notwendigen SQL Statements neu erstellt werden.

6 ANWENDUNGSSOFTWARE

Die Anwendungssoftware umfasst die Verbindung zur Steuerungstechnik sowie alle weiteren Funktionen des zu emulierenden Warehouse-Management-Systems. In diesem Kapitel soll zunächst auf die Architektur der Anwendung eingegangen, anschließend sollen die Anforderungen an die Anwendungsteile definiert und von diesen die notwendigen Klassen, Methoden und Eigenschaften abgeleitet werden.

6.1 Architektur

Die Architektur beschreibt die grundlegenden Komponenten der Anwendung und deren Zusammenspiel. In Zusammenarbeit mit den Entscheidungsträgern von Jungheinrich werden folgende zwei Anforderungen an die Architektur definiert:

1. Möglichst logischer Aufbau der Klassen
2. Die Anwendung soll sich beim Starten mit Hilfe der Konfigurationsdaten aus der Datenbank selbständig konfigurieren

Der Abbildung 39 kann entnommen werden, dass jede umfangreichere Funktion sowie jedes Element des Emulators von einer eigenen Klasse abgebildet wird. Diese Aufteilung soll so einem möglichst logischen Aufbau der Klassen folgen. Die Instanz der Main Klasse besitzt für jede Speicherprogrammierbare Steuerung der Anlage eine Instanz der Plc Klasse sowie eine der OrderManagement Klasse für alle vom Benutzer erstellten Aufträge. Die Plc Instanzen können aus mehreren Verbindungen und Meldepunkten bestehen. Jeder Meldepunkt hat eine eigene Instanz der Route Klasse, die alle Wegeberechnungen für diesen vornimmt. Mittels Vererbung können die Funktionen der Basisklassen möglichst allgemein bleiben und die weiteren vom Typ abhängigen Funktionen von der erbenden Klasse umgesetzt werden.

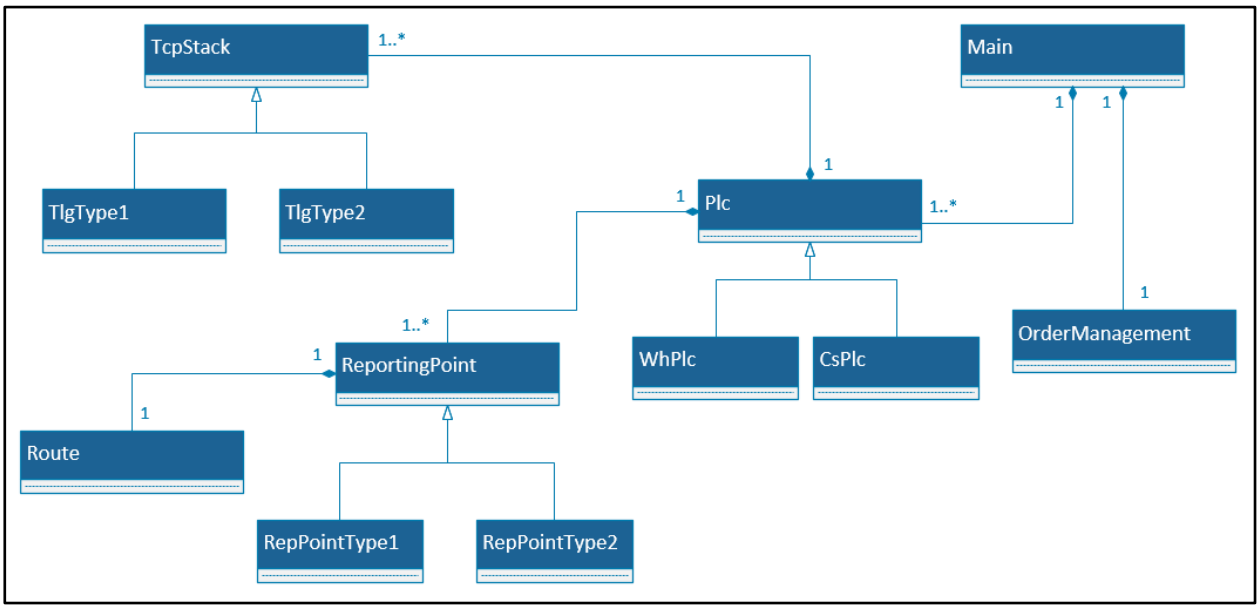


Abbildung 39: Architekturübersicht Anwendungssoftware, Quelle: Eigene Darstellung

Beim Hochlaufen des Emulators werden die Informationen über den Aufbau der Anlage aus der Datenbank geladen. Für jeden Eintrag in der Tabelle tblPlc wird eine Instanz der Klasse Plc im Main erstellt. Der Typ definiert dabei, ob es sich um eine Steuerung eines Regalbediengerätes oder um eine der Fördertechnik

handelt. Bei der Instanziierung dieser Klasse können dann die genaueren Daten zu den Meldepunkten der Steuerung sowie deren Verbindungen geladen werden. Mit ihrer Hilfe werden weiters auch deren Instanzen erstellt. Dieser Vorgang beim Starten ermöglicht in Verbindung mit der Nutzung von generischen Listen, dass die Anwendung des Emulators einfach an unterschiedliche Anlagen anpassbar ist und beim Hochlaufen vollständig konfiguriert wird.

6.2 Kommunikation zur Steuerungsebene

6.2.1 Anforderungen

Die Kommunikation mit der Steuerungsebene soll über TCP erfolgen. Der Telegrammverkehr kann sich von System zu System unterscheiden und ist daher getrennt vom TCP-Teil zu entwickeln. Folgende Variationen des Verbindungsaufbaus und Telegrammaustauschs müssen bei der Programmierung der Kommunikation bedacht werden:

Einkanalige Kommunikation

Der gesamte Telegrammverkehr zwischen Emulator und SPS erfolgt über einen Kanal.

Zweikanalige Kommunikation

Der Emulator und die SPS senden auf verschiedenen Kanälen, wobei die Telegramm-Quittung immer auf dem Kanal erwartet wird, auf dem das Telegramm gesendet wurde.

Mehrkanalige Kommunikation

Die Kommunikationskanäle können den Meldepunkten zugeordnet werden. So kann eine SPS über n-Kanäle mit dem Emulator kommunizieren, wobei ein Kanal n-Meldepunkten zugeordnet werden kann.

6.2.2 Lösungskonzept

Um die TCP Verbindung getrennt vom Telegrammverkehr entwickeln zu können, soll eine eigene TCP-Stack Klasse erstellt werden. Diese Klasse dient als Basisklasse der unterschiedlichen Telegrammklassen und soll sich nur um die Verbindung zur Steuerung sowie um das Empfangen und Senden von Daten kümmern. Das Format der Daten wird in dieser Klasse nicht berücksichtigt. Die Übergabe der Daten an die Telegrammklassen soll über eine virtuelle Methode der Basisklasse namens EvaluateTlg erfolgen, die von den erbenden Klassen überschrieben wird.

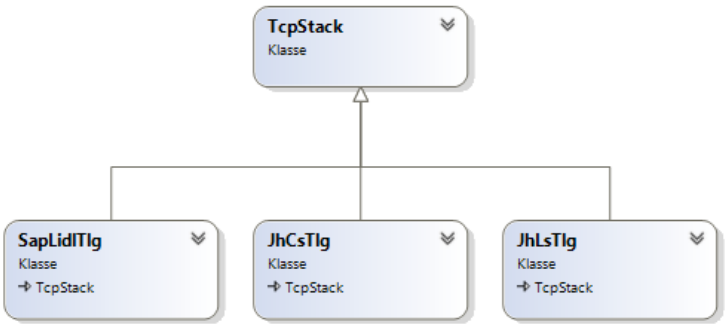


Abbildung 40: Kommunikations-Klassen, Quelle: Eigene Darstellung

Der Abbildung 40 können die Beziehungen zwischen den Klassen entnommen werden.

Für eine einkanalige Kommunikation, bei der alle Telegramme über den gleichen TCP Kanal gesendet werden, reicht es, die Telegrammklasse einmal zu instanziieren. Für jeden weiteren Kanal müssen die Klassen wiederholt instanziiert werden. Die Zuordnung der Verbindungen zu den Meldepunkten erfolgt über deren Beziehung in der Datenbank, in der auch die genauen Parameter sowie der Typ der Verbindung gespeichert sind.

6.3 Meldepunkte

Als Meldepunkt wird eine Position in der Anlage bezeichnet, an der Telegramme zwischen dem Emulator und der Steuerungsebene ausgetauscht werden. Meldepunkten können verschiedene Funktionen und Aufgaben zugeteilt werden, wodurch sich verschiedene Meldepunkttypen ergeben.

6.3.1 Anforderungen

Der Basismeldepunkt trifft Wegentscheidungen und dient der Lokalisierung von LHM. Folgende Funktionen und Eigenschaften sollen diesem zugeteilt werden:

- Ein Meldepunkt kann n-mögliche Ziele bzw. Routen haben
- Zu einem Meldepunkt dürfen maximal 1, n oder unendliche viele LHMs gesendet werden.
- Ein Meldepunkt kann verschiedene Status einnehmen wie:
    - **Ready**: Der Meldepunkt ist für weitere LHM bereit; dieser Zustand ist von seiner aktuellen Auslastung abhängig.
    - **Error**: Der Meldepunkt hat einen Fehler; es sollen keine weiteren LHM zu diesem Punkt gesendet werden.
    - **Locked**: Der Meldepunkt ist gesperrt; es dürfen keine weiteren LHM zu diesem Punkt gesendet werden.
    - **Hold**: Der Meldepunkt ist angehalten; es dürfen weitere LHM zu diesem Punkt gesendet, aber von dort aus nicht weitersenden werden. (Diese Funktion dient dem Puffern von LHM bei einem Leistungstest.)
- Einem Meldepunkt können begrenzende Eigenschaften, wie eine maximale Höhe oder Grundfläche, zugeordnet werden.

Neben dem Basismeldepunkt kann es weitere Typen mit speziellen Funktionen geben:

Identifikationspunkt:

Am Identifikationspunkt werden Wareneingangsprüfungen durchgeführt. Zusätzlich kann es an diesem Punkt anlagenspezifische Besonderheiten geben, wie das Zusammenfassen von Halbpaletten zu einer Ganzpalette.

Abnahmepunkt (A-Punkt):

Am Abnahmepunkt verlässt das LHM die Anlage. Wird das LHM vom A-Punkt abgenommen, wird es aus dem System entfernt.

Einlagerungspunkt:

Der Einlagerungspunkt dient der Schnittstelle zwischen Fördertechnik und Hochregallager. Spätestens an diesem Punkt muss ein freier Platz in der Gasse gesucht werden. Neben der Lagerplatzsuche soll dieser Punkt Einlageraufträge für das RBG erzeugen können.

Auslagerungspunkt:

Der Auslagerungspunkt dient wie der Einlagerungspunkt der Schnittstelle zwischen Fördertechnik und Hochregallager. LHM, die vom RBG ausgelagert werden, können an dieser Stelle an die Fördertechnik übergeben werden. Der Auslagerungspunkt soll notwendige Sequenzierungsinformationen an die Steuerungsebene weitergeben können.

Lastaufnahmemittel (LAM):

Das Lastaufnahmemittel ist eine Position am RBG. Ein RBG kann mehrere LAM haben, wobei diese in verschiedensten physikalischen Anordnungen am RBG angebracht sein können

6.3.2 Lösungskonzept

Für den Basismeldepunkt soll eine Klasse namens ReportingPoint erstellt werden, die alle Methoden und Eigenschaften eines Standardmeldepunkts beinhaltet. Alle weiteren Typen sollen von dieser Klasse erben und wenn nötig die einzelnen Methoden überschreiben können.

Die wichtigsten Methoden des Basismeldepunktes sind LhmArrived und LhmLeft. Diese als virtuell gekennzeichneten Methoden werden aufgerufen, sobald ein LHM am Meldepunkt ankommt beziehungsweise diesen wieder verlässt.

Methode – LhmArrived:

Signalisiert die Steuerungsebene durch ein Telegramm, dass ein LHM am Meldepunkt eingetroffen ist, ändert diese Methode den Ort des Ladehilfsmittels. Zusätzlich wird versucht, ein neues Ziel zu suchen und die Route dorthin zu bestimmen.

Methode – LhmLeft:

Wurde ein neues Ziel für das Ladehilfsmittel gefunden, kann die aktuelle Anzahl der LHM an diesem Punkt verringert und die Anzahl am Zielmeldepunkt erhöht werden. Der Ort des LHM wird dabei wieder aktualisiert. Nach dem Anpassen der Daten in der Datenbank wird der neue Transportauftrag an die Steuerung gesendet.

Eigenschaft – DbEntry

Diese Eigenschaft vom Typ tblReportingPoint enthält alle aktuellen Daten des Meldepunktes aus der Datenbank. Die Änderung dieser Eigenschaft führt zur

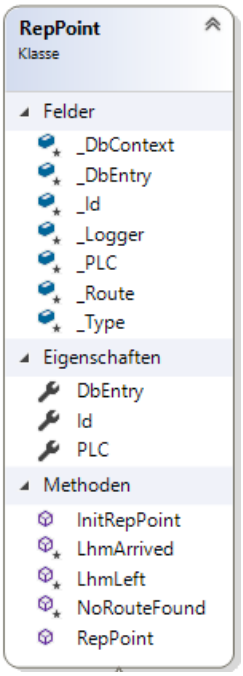


Abbildung 41: Klasse RepPoint, Quelle: Eigene Darstellung

Aktualisierung des Tabelleneintrags und ermöglicht, dass alle weiteren Instanzen der Anwendung automatisch immer die aktuellsten Daten erhalten.

Der Abbildung 42 können die einzelnen Meldepunktklassen entnommen werden. Wie zuvor erwähnt erbt jeder Typ vom Basismeldepunkt und kann damit dessen Methoden und Eigenschaften erweitern.

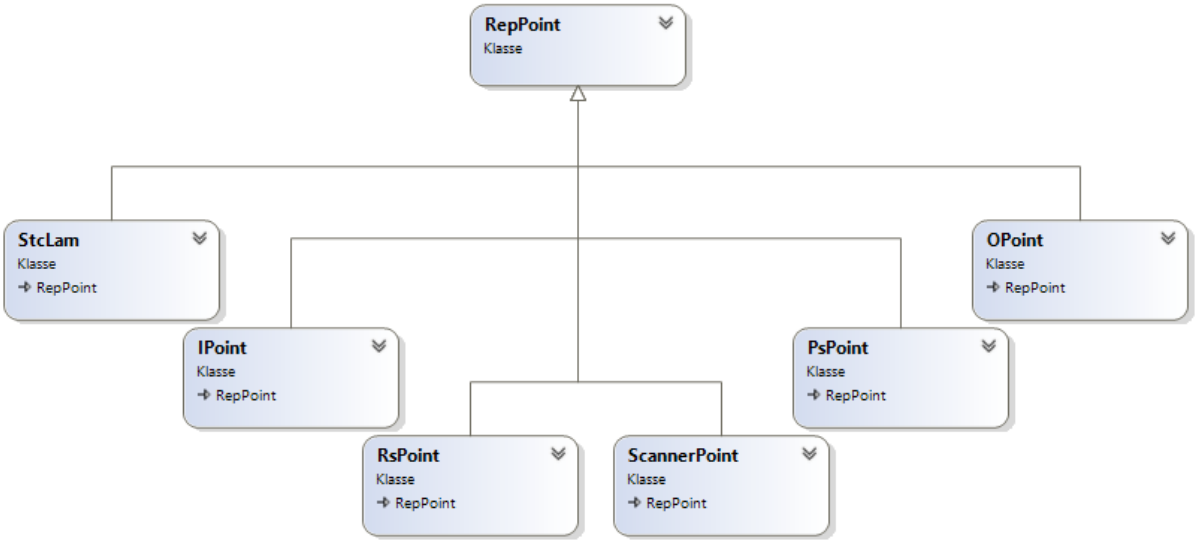


Abbildung 42: Meldepunkt - Klassen, Quelle: Eigene Darstellung

Identifikationspunkt (IPoint):

Wird der Emulator über ein neues LHM am Identifikationspunkt informiert, soll dieses mittels CheckLhm Methode geprüft werden. Dabei werden der Barcode auf seine Eindeutigkeit sowie die Höhe und das Gewicht des LHM geprüft. Verläuft die Prüfung erfolgreich, kann ein neuer Eintrag für das Ladehilfsmittel in der Datenbank erstellt und ein neues Ziel gesucht werden. Tritt bei der Wareneingangsprüfung ein Fehler auf, wird als neues Ziel des Ladehilfsmittels der nächste NiO (Nicht in Ordnung) Punkt festgelegt. Ein neuer Datensatz wird dabei nicht erstellt.

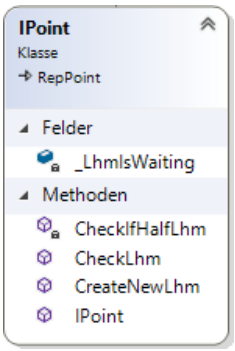


Abbildung 43: Klasse IPoint, Quelle: Eigene Darstellung

Abnahmepunkt (OPoint):

Wird der Emulator über das Abnehmen eines LHM am Abnahmepunkt informiert, so kann dieses mittels DeleteLhm Methode aus der Datenbank gelöscht werden.

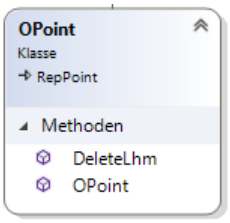


Abbildung 44: Klasse OPoint, Quelle: Eigene Darstellung

Einlagerungspunkt (PsPoint):

Die PsPoint Klasse überschreibt die Methode LhmArrived der Basisklasse ReportingPoint. Trifft ein LHM an diesem Punkt ein, soll nicht nur der Ort geändert und ein neues Ziel gesucht, sondern auch ein Einlagerungsauftrag für das Regalbediengerät erstellt werden. Dafür wird ein neuer Eintrag in der Tabelle tblVehicleOrder in der Datenbank angelegt. Als Quelle wird der Einlagerungspunkt selbst definiert, als Ziel das Ergebnis der zuvor angestoßenen Lagerplatzsuche.

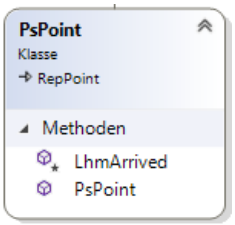


Abbildung 45: Klasse PsPoint, Quelle: Eigene Darstellung

Auslagerungspunkt (RsPoint):

Die RsPoint Klasse überschreibt wie schon die PsKlasse die Methode LhmArrived. Zu den Basisfunktionen der LhmArrived Klasse können, wenn notwendig, Funktionen zur Sequenzierung hinzugefügt werden. Ein Beispiel dafür wäre das Informieren der Steuerungsebene über das letzte Ladehilfsmittel eines Auslagerauftrages.

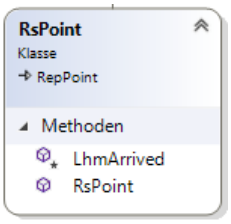


Abbildung 46: Klasse RsPoint, Quelle: Eigene Darstellung

Lastaufnahmemittel (StcLam):

Das Lastaufnahmemittel (LAM) ist eine Sonderform des Meldepunktes, da es sich in seinen Funktionen stark von den anderen Meldepunkten unterscheidet und dessen Aufträge von der übergeordneten WhPlc Klasse bearbeitet werden. Wird von der WhPlc Klasse ein neuer Auftrag aus der Tabelle tblVehicleOrder gestartet, so wird er zunächst einem Lastaufnahmemittel durch Aufrufen der Methode PickUp, Deposit oder PickUpDeposit zugeordnet. Die StcLam Instanz speichert sich die Identifikationsnummer des Auftrages und sendet je nach Methode den gesamten Transportauftrag oder nur dessen Aufnahme oder Abgabe an das Regalbediengerät. Wird der Auftrag von der Steuerungsebene als beendet gemeldet, wird die Methode PickUpDone oder DepositDone aufgerufen, die den Status des Auftrages in der Datenbank aktualisiert.

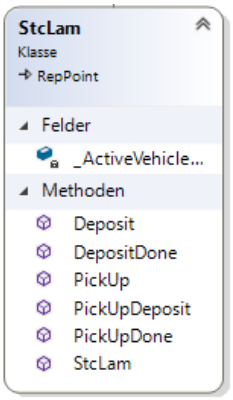


Abbildung 47: Klasse StcLam, Quelle: Eigene Darstellung

Die Zuordnung der Aufträge zu den Lastaufnahmemitteln erfolgt von der WhPlc Klasse auf Basis der möglichen Routen der einzelnen LAM. Nicht jedes LAM kann auf Grund der mechanischen Gegebenheiten des Regalbediengeräts jeden Auftrag ausführen. Der Abbildung 48 kann zum Beispiel entnommen werden, dass das LAM2 nicht das LHM1 aufnehmen kann. Die Aufnahme beider LHM ist somit nur möglich, wenn LHM1 von LAM1 aufgenommen wird und LHM2 von LAM2.

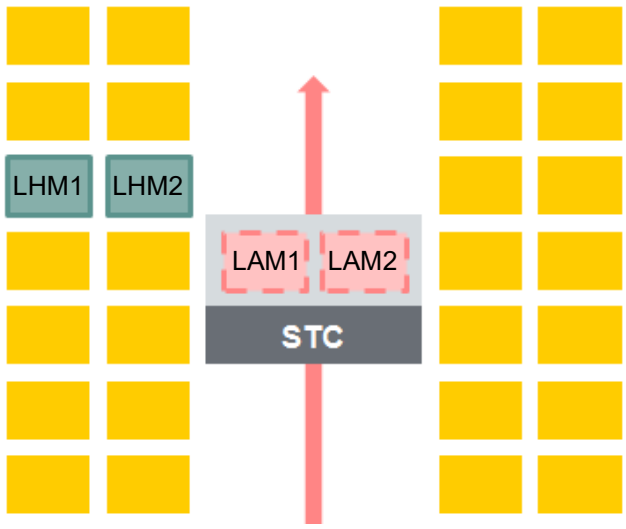


Abbildung 48: Anordnung der Lastaufnahmemittel auf einem Regalbediengerät, Quelle: Eigene Darstellung

6.4 Routen / Lagerplatzsuche

Als Route wird der Weg zwischen zwei Meldepunkten bezeichnet. Mehrere zusammengefasste Routen werden Wegstrecke genannt. Bei der Lagerplatzsuche wird ein freier Platz im Hochregallager gesucht; die dafür benötigten Funktionen ähneln denen der Routenbildung. Aus diesem Grund werden beide Funktionen in diesem Kapitel zusammengefasst.

6.4.1 Anforderungen

Ein Meldepunkt kann wie zuvor erwähnt n Routen haben. Für die Auswahl der geeigneten Wegstrecke sind einer Route verschiedene Eigenschaften zuzuordnen wie:

- Länge der Route
- Maximale Höhe, Grundfläche, Gewicht
- Status der Zielmeldepunkte der Route
- Weitere Sonderfälle

Bei der Lagerplatzsuche wird ein freier Lagerplatz im Hochregallager gesucht. Für die Auswahl des Platzes müssen folgende Kriterien eingehalten werden:

- Alle Gassen sollen gleichmäßig befüllt werden
- Die Höhe und Breite des Regalfaches muss mit der des LHMs übereinstimmen
- Auffüllen des Lagers von unten (Stabilität des Lagers)
- LHM dürfen nur auf freien und nicht gesperrten Lagerfächern abgestellt werden

6.4.2 Lösungskonzept

Um die in Kapitel 6.4.1 definierten Anforderungen erfüllen zu können, soll eine Klasse mit verschiedenen Methoden und Eigenschaften entwickelt werden. Die zu entwickelnde Klasse soll die optimale Route zu einem Wunschziel, das optimale Ziel mit Route oder den am besten geeignetsten Lagerplatz suchen können.

Methode – SearchBestTarget

Ziel dieser Methode ist es, das geeignetste und nächstgelegene Ziel für ein Ladehilfsmittel auszuwählen. Die Wegstrecke zu diesem Ziel darf dabei wie in Abbildung 49 nur aus einer einzigen Route bestehen. Der Übergabewert der Methode ist der Datenbankeintrag des LHM. Als Rückgabewert wurde der Datenbankeintrag der Route definiert.

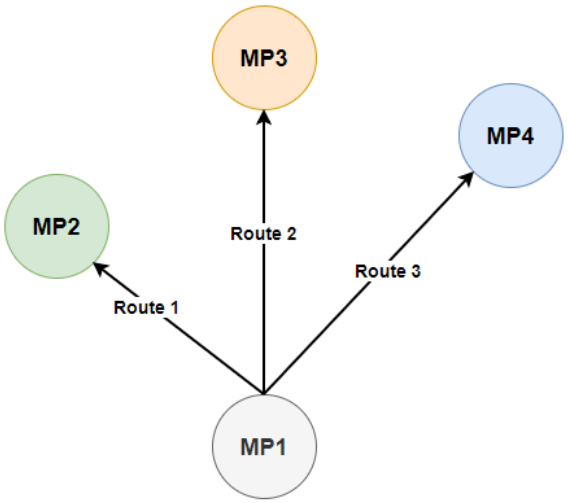


Abbildung 49: Zielsuche Klasse Route, Quelle: Eigene Darstellung

Die Auswahl des Ziels wird aufgrund von folgenden Kriterien getroffen:

1. Ist dem Ladehilfsmittel eine Routenbedingung zugeordnet, so muss die gewählte Route dieser Bedingung entsprechen. Ein Beispiel wäre etwa ein fehlerhafter Barcode oder jeglicher anderer projektbezogener Sonderfall.
2. Weder das gewählte Ziel noch die Route dorthin darf im Status Locked sein.
3. Die Größe des Ladehilfsmittels muss kleiner als das maximale Abmaß des Meldepunktes oder der Route sein.
4. Meldepunkte mit einer geringeren Auslastung sollen bevorzugt behandelt werden.
5. Treffen die zuvor genannten Kriterien auf mehr als einen Meldepunkt zu, so wird zufällig ein Meldepunkt ausgewählt.

Methode – SearchBestRoute

Ist das Endziel des Ladehilfsmittels bekannt, so kann mit Hilfe dieser Funktion die kürzeste Wegstrecke dorthin bestimmt werden. Die Länge der möglichen Wegstrecken ergibt sich aus der Summe aller Streckenlängen der Routen, die vom Quell- zum Zielmeldepunkt führen. Als Übergabeparameter gilt der Datenbankeintrag des zu bewegenden LHM – das Endziel ist in diesem Datensatz bereits enthalten.

Für die Berechnung des kürzesten Weges soll der Algorithmus von Edsger Wybe Dijkstra angewendet sowie alle Kriterien der Methode SearchBestTarget berücksichtigt werden. Dazu werden vor der Anwendung des Algorithmus alle Streckenlängen der gesperrten Routen sowie aller Routen, die zu einem gesperrten Ziel führen, mit einer unendlichen Länge überschrieben.

Methode – SearchStorageLocation

Die Funktion dieser Methode ähnelt der SearchBestTarget Funktion. Ziel dieser Methode ist es auch, ein möglichst ideales Ziel für ein LHM zu suchen. Der Unterschied wird in den folgenden Kriterien der Suche sichtbar.

1. Im ersten Schritt werden alle gesperrten sowie zu kleinen Lagerplätze bzw. Routen herausgefiltert.
2. Danach wird, wenn die Wahl zwischen mehreren Lagergassen besteht, versucht, eine Gleichverteilung der Lagerauslastung zu erreichen.
3. Nach der Wahl der Lagergasse werden die noch möglichen Stellplätze nach der Ebene im Lager sortiert.
4. Zwischen den übrigen Stellplätzen auf der gleichen Ebene wird wieder zufällig entschieden.

Wird für ein LHM mit Höhe 2 ein Lagerplatz in einem Lager wie in Abbildung 50 gesucht, so soll zwischen den dunkelgrün markierten Lagerplätzen zufällig gewählt werden.

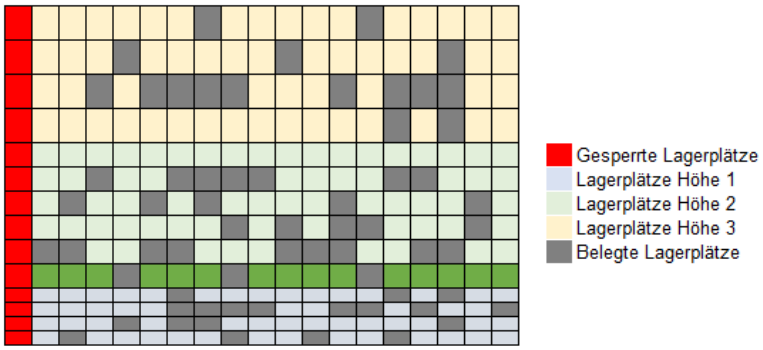


Abbildung 50: Lagerplatzsuche, Quelle: Eigene Darstellung

6.5 Auslageraufträge

Um Ladehilfsmittel vom Hochregallager zu den Versandbahnen oder Kommissionier-Plätzen senden zu können, benötigt es Auslageraufträge. Diese Aufträge müssen händisch über die Benutzeroberfläche angelegt werden. Ziel dieses Unterkapitels ist es, Anforderungen an das Erstellen und das Abarbeiten dieser Aufträge zu definieren sowie ein Lösungskonzept zu entwickeln.

6.5.1 Anforderungen

Das Erstellen von Auslageraufträgen soll über die Benutzeroberfläche angestoßen werden. Dabei müssen die Anzahl an LHM und das Auslagerungsziel definiert werden. Die Auswahl der auszulagernden LHM kann automatisch, mit der Bestimmung die ältesten Paletten zuerst, oder manuell über eine Auflistung aller vorhandenen LHM erfolgen. Zusätzlich soll es möglich sein, nur LHM aus einer bestimmten Gasse auszulagern.

Nach dem Anlegen der Auslageraufträge müssen die Aufträge den Regalbediengeräten zugeordnet und in einzelne Fahraufträge umgewandelt werden.

6.5.2 Lösungskonzept

Das Erstellen der Aufträge erfolgt wie zuvor erwähnt über die Benutzeroberfläche – demnach sollen auch alle Filterfunktionen von der Oberfläche umgesetzt werden. Die Anwendungssoftware übernimmt erst, nachdem die Auslageraufträge in der Datenbanktabelle tblOrder angelegt sind.

Nach dem Anlegen eines neuen Auftrages werden die Aufträge von der OrderMgmt Klasse zunächst gassenweise vorsortiert. Nach der Aufteilung auf die jeweilige Gasse werden die Wegstrecken zum Auslagerziel bestimmt, um anschließend die Transportaufträge für die Regalbediengeräte erstellen zu können. Die Wegstrecke vom Lagerplatz zur Versandbahn OP1 mit dem Zwischenziel RS1 (Auslagerungsstation) wird in der Abbildung 51 grafisch dargestellt.

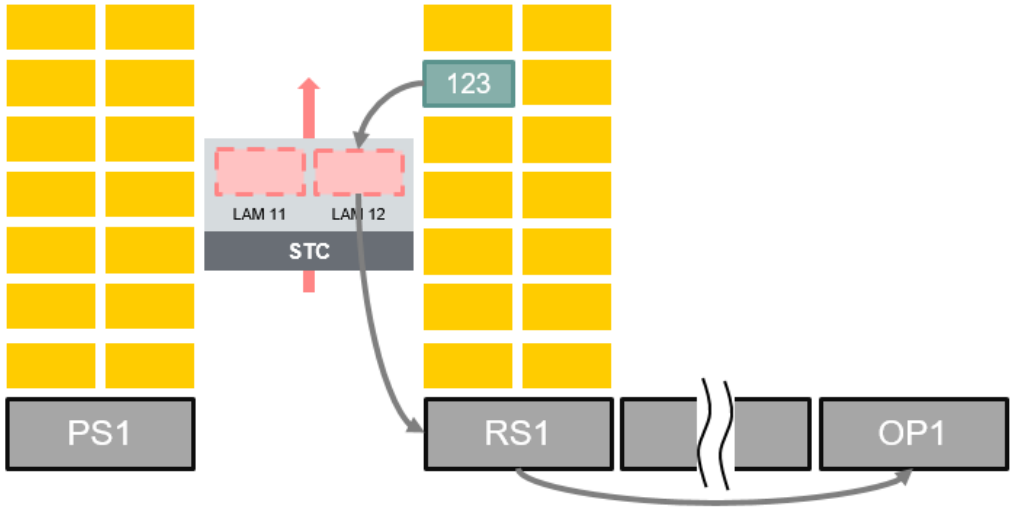


Abbildung 51: Wegstrecke vom Lagerplatz zur Versandbahn, Quelle: Eigene Darstellung

7 BENUTZEROBERFLÄCHE

Die Oberfläche ermöglicht dem Benutzer des Emulators, Funktionen auszulösen und Informationen über dessen aktuellen Zustand zu erhalten. Ziel dieses Kapitels ist es, die Anforderungen an die Oberfläche in Teilschritten zu definieren und dafür ein Lösungskonzept zu entwickeln. Die einzelnen Teilschritte sind das Design der Oberfläche, die auszulösenden Funktionen und das MVVM Architekturmuster in der praktischen Umsetzung.

7.1 Design

Für die Benutzeroberfläche soll ein intuitives Design entwickelt werden, das dem des Jungheinrich-WMS ähnelt, um den Bedienern des Emulators einen einfachen Umstieg vom realen WMS zum Emulator zu ermöglichen. Das Hauptfenster sowie alle Dialoge sollen in den Jungheinrich-Farben gestaltet werden.

7.1.1 Hauptfenster

Das Hauptfenster dient der Navigation zwischen den einzelnen Seiten der Oberfläche und ist wie in Abbildung 52 ersichtlich in drei Teile geteilt:

1. **Navigationsbereich**
   Dieser Bereich ist mit einem Hamburger Menu ausgestattet und ermöglicht das Wechseln zwischen den im Inhaltsbereich angezeigten Seiten.
2. **Inhaltsbereich**
   In diesem Bereich werden die Inhalte der Seite angezeigt, die über das Hamburger Menu ausgewählt wurde. Die Seiten selbst folgen keinem strikten Designmuster, außer dass die Jungheinrich-Farben für Grafiken verwenden werden sollen.
3. **Statuszeile**
   In der Statuszeile können allgemeine Informationen über den Status des Emulators angezeigt werden.

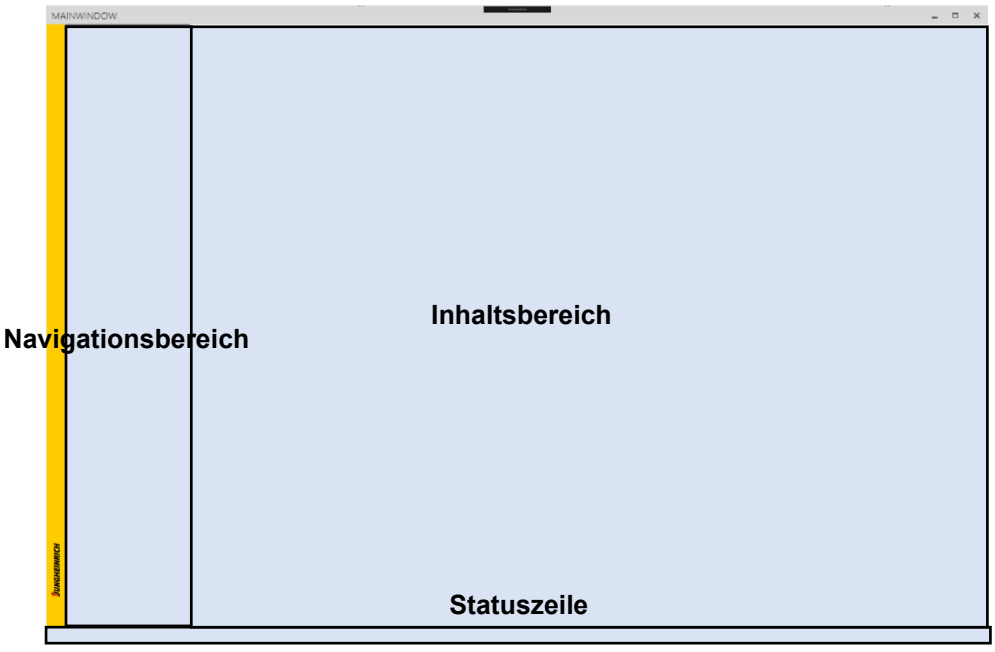


Abbildung 52: Aufteilung des Hauptfensters, Quelle: Eigene Darstellung

7.1.2 Begrüßungsbildschirm

Der Begrüßungsbildschirm (Splash) soll Informationen über die aktuelle Version des Emulators liefern und beim Start der Anwendung angezeigt werden. Das Design wurde zum Teil von dem des Jungheinrich-WMS übernommen. Die Darstellung des Splash soll wie in Abbildung 53 gestaltet werden.

Abbildung 53: Begrüßungsbildschirm WMS Emulator, Quelle: Eigene Darstellung

7.1.3 Dialoge

Das Design der Dialoge soll an das des Hauptfensters erinnern. Wie Abbildung 54 zeigt, findet sich der im Jungheinrich-Gelb gestaltete Balken auf der linken Seite des Fensters auch in den Dialogen wieder. Alle Dialoge sollen mit einem „Ok“-Button zum Bestätigen und wenn gewünscht einem „Cancel“-Button zum Abbrechen der Eingabe ausgestattet sein.

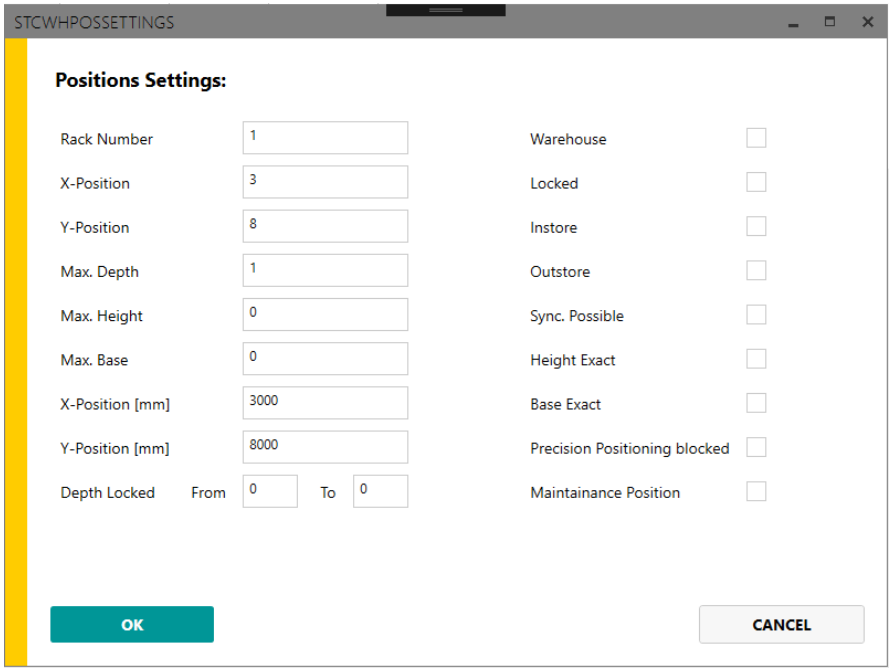


Abbildung 54: Beispiel für das Design eines Eingabedialoges, Quelle: Eigene Darstellung

7.2 Auszulösende Funktionen / anzuzeigende Informationen

Die auszulösenden Funktionen sowie alle anzuzeigenden Informationen sind ein wichtiger Teil des Konzepts der Benutzeroberfläche. Ziel dieses Unterkapitels ist es, diese zu definieren und ein Lösungskonzept zu entwickeln, wie sie an unterschiedliche Anlagen automatisch angepasst werden können.

- **Verbindungen zur Steuerungsebene**
  Der Status aller Verbindungen mit der Steuerungsebene soll auf einer Seite angezeigt und bei Bedarf getrennt und aufgebaut werden können. Zusätzlich soll es die Möglichkeit geben, Parameter der Verbindungen wie IP-Adressen abzuändern. Eine weitere Funktion auf dieser Seite ist das manuelle Senden von Telegrammen, mittels derer Nachrichten selbständig vom Bediener zusammengestellt und zur Steuerung gesendet werden können.
- **Suchen von Ladeeinheiten**
  Alle Ladeeinheiten, die sich in der Anlage befinden, sollen auf einer eigenen Seite angezeigt werden. Filtermöglichkeiten wie der aktuelle Ort, Erstellzeitpunkt, Barcode oder Ähnliches grenzen die Suche dabei ein.
- **Meldepunkte**
  Der Status sowie die aktuelle Kapazität jedes Meldepunktes der Fördertechnik sollen angezeigt werden können. Zusätzlich soll es die Möglichkeit geben, jeden Meldepunkt zu sperren oder in den Zustand „Hold“ zu bringen, sowie die maximale Kapazität des Punktes zu verändern. Über einen „Detail“-Button können außerdem die letzten 100 Bewegungen des Meldepunktes angezeigt werden.
- **Routen**
  Ähnlich wie bei den Meldepunkten soll auch auf dieser Seite der aktuelle Status der Routen abgefragt werden können. Neben der Anzeige des aktuellen Status soll es ebenfalls möglich sein, jede Route einzeln zu sperren sowie deren Eigenschaften zu verändern. Zusätzlich wird die Anzahl aller über die Routen gefahrenen Ladehilfsmittel dargestellt.
- **Lagerübersicht**
  In der Lagerübersicht sollen der Status aller Lastaufnahmemittel sowie aller Lagerfächer visualisiert werden. Der Zustand der LAM kann dabei in einer Liste angezeigt werden, während der Zustand der Lagerfächer in einem Raster grafisch dargestellt wird. In diesem Raster soll jede Zelle einem Lagerfach zugeordnet werden – die verschiedenen Farben der Zellen geben dabei Aufschluss über deren Status und Kapazität. Verschiedene Registerkarten ermöglichen das Wechseln zwischen den Lagerzeilen. Zudem sollen die Lagerfächer einzeln gesperrt werden können.
- **Auslagerungsaufträge**
  Wie bereits in Kapitel 6.5 erwähnt sollen Auslagerungsaufträge über die Benutzeroberfläche erstellt werden können. Die Aufträge werden im ersten Schritt erstellt, wobei ausgewählt werden kann, ob die Ladehilfsmittel aus allen Gassen ausgelagert werden dürfen oder nur aus bestimmten und wie viele LHM betroffen sind. Nach dem Erstellen der Aufträge soll diese Seite den Zustand der Aufträge sowie deren Fortschritt anzeigen. Zusätzlich müssen die Aufträge einzeln gestartet, gestoppt und gelöscht werden können.

7.3 Automatische Konfigurierung der Bedienoberfläche

Wie schon die Anwendung des Emulators selbst soll auch die Bedienoberfläche so konzipiert sein, dass so wenig projektbezogene Anpassungen wie möglich vorgenommen werden müssen. Um weiters möglichst wenig Code im ViewModel entwickeln zu müssen und so die Übersichtlichkeit zu bewahren, werden bis auf die Lageransicht keinerlei Steuerelemente im C# Code bei Laufzeit erstellt.

Um diese Anforderungen erfüllen zu können, sollen bei der Entwicklung des Bedienoberfläche hauptsächlich Tabellen und Dropdown-Felder verwendet werden, die beim Starten der Anwendung befüllt werden.

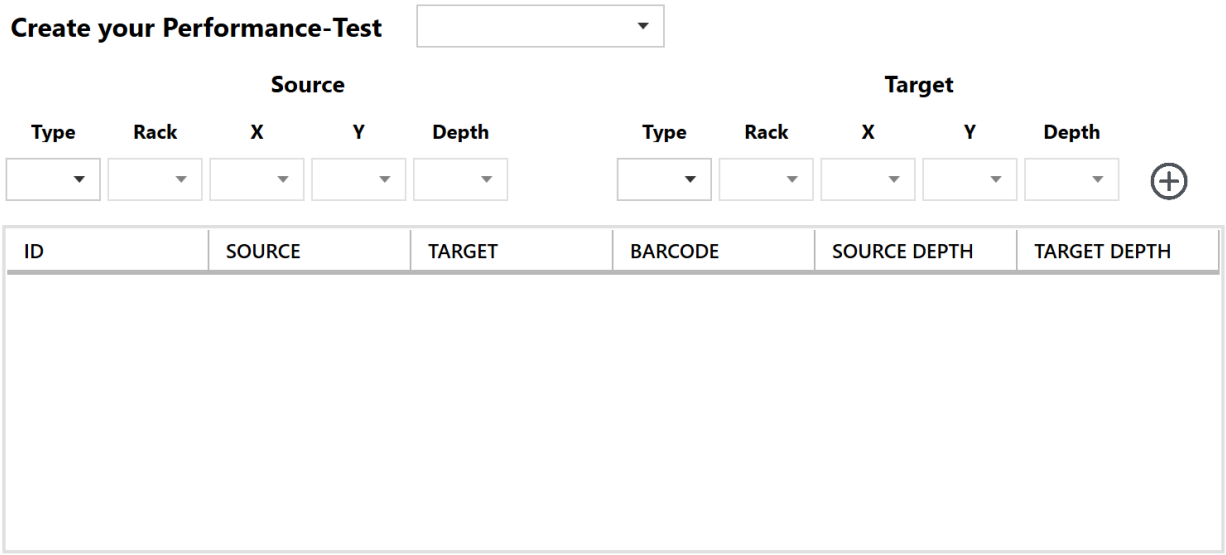


Abbildung 55: Einsatz von Tabellen und Dropdown Feldern in der Benutzeroberfläche, Quelle: Eigene Darstellung

7.4 Model View ViewModel in der praktischen Umsetzung

Der theoretische Teil des Model View ViewModel (MVVM) Architekturmusters wurde bereits im Kapitel 4.5.1 behandelt. Inhalt dieses Unterkapitels ist die praktische Verwendung dieses Musters bei der Entwicklung der Benutzeroberfläche des Emulators.

Wie im theoretischen Teil dieser Arbeit ausgeführt wurde, trennt das MVVM die Darstellung von der Logik. Diese Trennung führt dazu, dass alle Fenster und Seiten der Oberfläche aus drei Teilen bestehen, der View, dem ViewModel und dem Model. Das Model jedes Fensters ist die Datenbank, da dort alle anzuzeigenden Daten gespeichert sind. Die View enthält das deklarative XAML-Markup mit allen Steuerelementen – dieser Teil ist der, der vom Benutzer gesehen wird. Die Schicht zwischen dem Model und der View ist das ViewModel. Dieser Teil besteht aus C# Code und beinhaltet die Logik der Anwendung. Weiters wird die View zu den Eigenschaften des ViewModels gebunden. Die Aufteilung der drei Schichten wird in Abbildung 56 grafisch dargestellt.

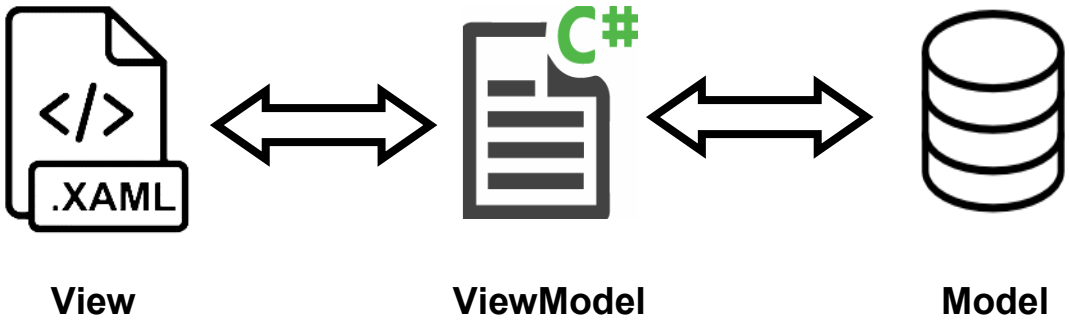


Abbildung 56: Teile des MVVM, Quelle: Eigene Darstellung

Im Kapitel 7.3 wurde bereits definiert, dass für die automatische Konfiguration der Bedienoberfläche hauptsächlich Tabellen und Dropdown-Felder verwendet werden sollen. Die Verwendung dieser Steuerelemente unter Zuhilfenahme des MVVM soll anhand von zwei Beispielen erläutert werden:

Tabelle (Datagrid)

Ein Datagrid ermöglicht es, Daten in einem veränderbaren Raster im View anzuzeigen. Die Bindung an die anzuzeigenden Daten des ViewModel wird dabei über die Eigenschaft „ItemSource“ möglich. Die Erstellung des Datagrid mit der Bindung an das ViewModel erfolgt in vier Schritten:

1. Erstellen des Datagrids im XAML-Markup der View
2. Erstellen der Datacollection Eigenschaft im ViewModel
3. Binden der ItemSource Eigenschaft des Datagrids an die Datacollection des ViewModels
4. Verbinden des DataContext der View mit dem ViewModel.

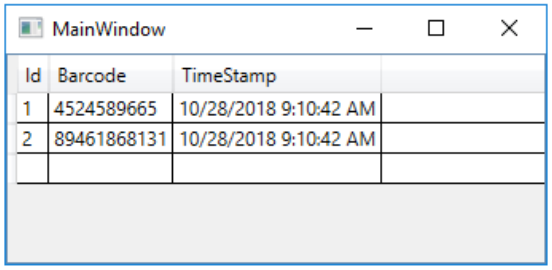


Abbildung 57: Datagrid in WPF, Quelle: Eigene Darstellung

Dropdown Feld (Combobox)

Eine Combobox gibt dem Benutzer die Möglichkeit, zwischen mehreren Elementen zu wählen. Ein Beispiel wäre die Wahl aus verschiedenen Versandbahnen bei der Erstellung eines Auslagerungsauftrages. Wie schon beim Datagrid kann auch die ItemSource Eigenschaft der Combobox an eine DataCollection, eine Enumeration oder ein Array gebunden werden. Um das gewählte Element der Combobox erkennen zu können, soll auch die SelectedIndex Eigenschaft der Combobox an eine Eigenschaft des ViewModels gehängt werden. Das Erstellen der Combobox erfolgt in fünf Schritten:

1. Erstellen der Combobox im XAML-Markup der View
2. Erstellen der Eigenschaften im View Model
3. Binden der ItemSource Eigenschaft der Combobox an ein Array of String des ViewModels

4. Binden der SelectedIndex Eigenschaft der Combobox an einen Integer Wert des ViewModels
5. Verbinden des DataContext der View mit dem ViewModel.

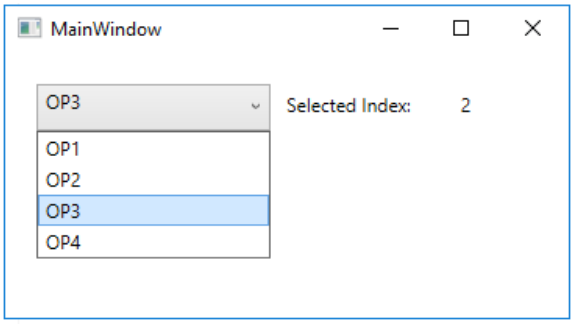


Abbildung 58: Combobox in WPF, Quelle: Eigene Darstellung

Neben dem Binden von Eigenschaften der View an Eigenschaften des ViewModels sind auch Kommandos (Commands) ein wichtiger Teil des MVVM. Commands werden immer dann benötigt, wenn Ereignisse der View ausgelöst werden sollen. Ein Beispiel ist das Klicken auf einen Button.

Kommandos (Commands)

Commands ermöglichen es, Funktionen über die Bedienoberfläche auszulösen. Dazu muss die Command Eigenschaft des Steuerelements an eine Eigenschaft des Typs ICommand aus dem ViewModel gebunden werden. Um dieser Eigenschaft des ViewModels eine Methode zuweisen zu können, die beim Auslösen des Commands aufgerufen wird, muss eine neue Klasse mit dem Namen RelayCommand erstellt werden.

Die Klasse RelayCommand erbt vom Interface ICommand und überschreibt dessen Methoden und Ereignisse. Beim Instanziieren der Klasse wird als Übergabewert die Methode angegeben, die durch die Methode Execute beim Auslösen des Commands aufgerufen wird. Der Abbildung 59 können die Felder, Methoden und Ereignisse der Klasse entnommen werden.

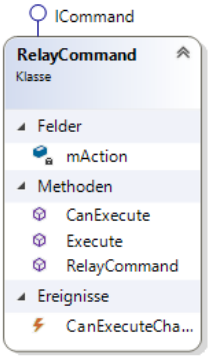


Abbildung 59: Klasse RelayCommand, Quelle: Eigene Darstellung

Das Erstellen eines Commands erfolgt in 6 Schritten und wird kurz anhand eines Buttons erklärt:

1. Erstellen des Buttons im XAML-Markup der View
2. Erstellen der ICommand Eigenschaften im View Model
3. Erstellen der vom Button aufzurufenden Methode
4. Instanziieren der ICommand Eigenschaft
5. Binden der Command Eigenschaft des Buttons an die ICommand Eigenschaft des ViewModels
6. Verbinden des DataContext der View mit dem ViewModel.

Die Abbildung 60 zeigt, wie durch den Klick auf den Button das Kommando ausgelöst wird, das eine MessageBox mit dem Text „You clicked the button“ angezeigt.

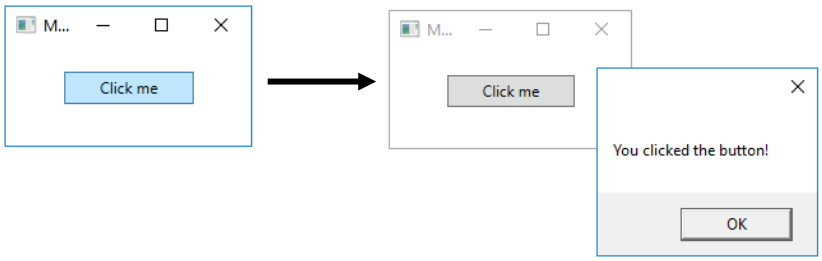


Abbildung 60: Click Command WPF, Quelle: Eigene Darstellung

Wechseln des Inhaltsbereiches im Hauptfenster

Um den Inhaltsbereich des Hauptfensters variabel zu gestalten und so zwischen den einzelnen Seiten der Benutzeroberfläche wechseln zu können, wird ein Hamburger Menu eingesetzt. Dieses Steuerelement ermöglicht es, verschiedene Seiten durch Klicken der Menu Items aufzurufen.

Jede einzelne Seite der Bedienoberfläche besteht wie am Beginn des Kapitels erwähnt aus einer View und einem ViewModel. Jede View hat eine Content-Eigenschaft, die über das ViewModel der Seite an das Hauptfenster weitergegeben werden kann. Im Hauptfenster kann dann über die Commands der Menu Items der richtige Content im Inhaltsbereich angezeigt werden. Das Anzeigen des Contents geschieht über das Binden der Content-Eigenschaft des Hamburger Menus zu einer Eigenschaft des Hauptfenster ViewModels vom Typ object. Diese Eigenschaft vom Typ object wird bei Auslösen der Menu Item Commands mit dem Content der Seite beschrieben. Die Abbildung 61 soll diesen Vorgang grafisch zeigen.

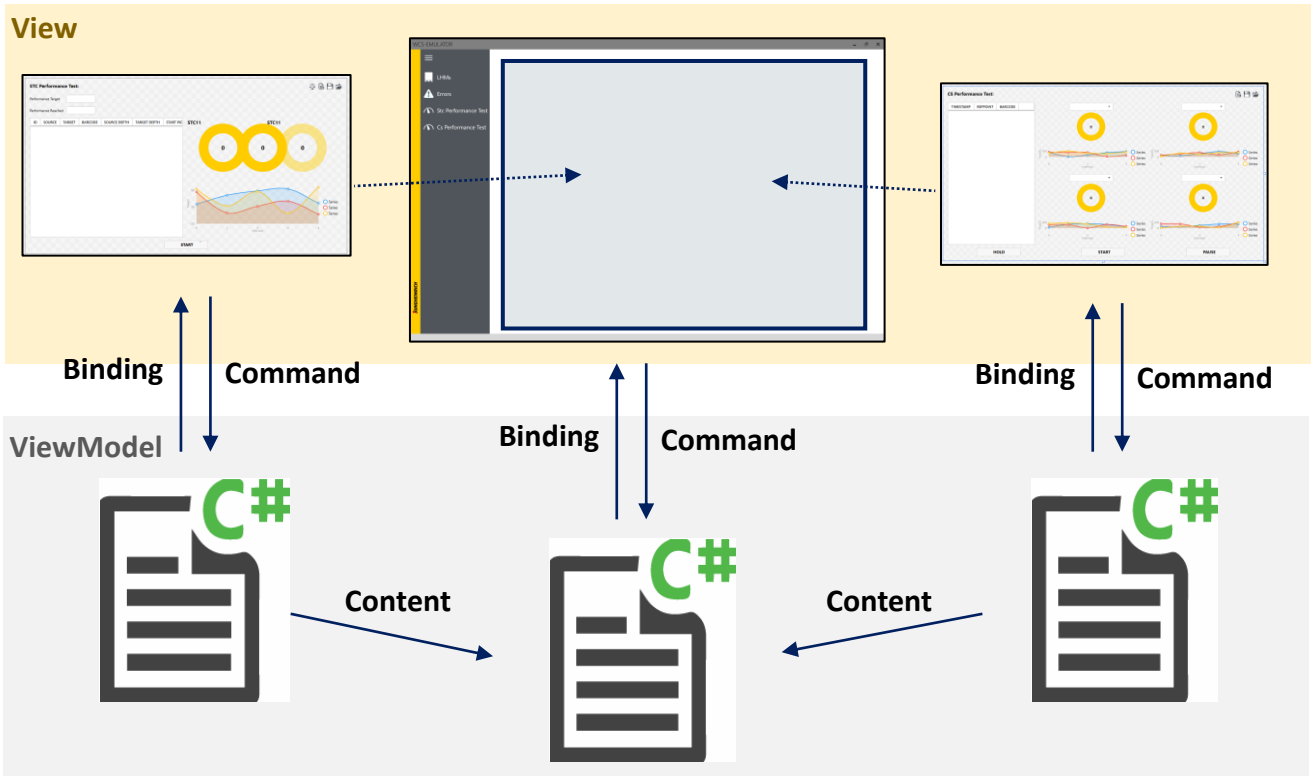


Abbildung 61: Wechesln des Inhaltsbereiches des Hauptfensters, Quelle; Eigene Darstellung

Early Prototyping

Vor dem Entwicklungsstart der Benutzeroberfläche sollen noch genauere Informationen über das Verhalten und die Funktionsweise der WPF-Bindungen gesammelt werden. Diese Informationen sollen durch Early Prototyping generiert werden.

Der Versuch soll Informationen über das Verhalten der Steuerelemente der View bei Änderung der Eigenschaften des ViewModels, zu der sie gebunden sind, liefern. Dazu wird ein DataGrid zu einer Liste des ViewModels gebunden. Nach dem Start der Anwendung soll durch einen Button ein neuer Datensatz zur Liste hinzugefügt werden. Der Versuch soll zeigen, ob sich das DataGrid automatisch aktualisiert oder ob dafür ein weiterer Schritt notwendig ist.

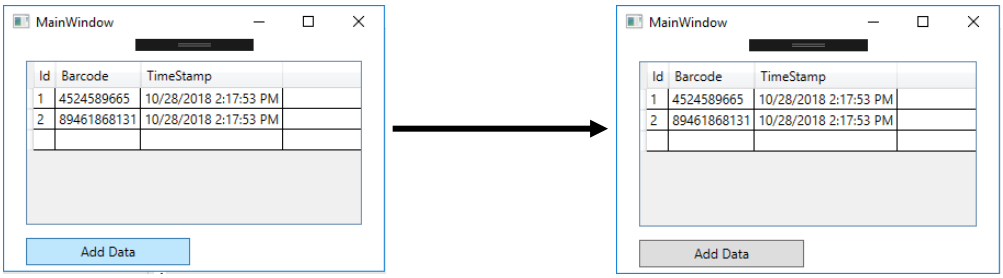


Abbildung 62: Versuch 1 Early Prototyping Benutzeroberfläche, Quelle: Eigene Darstellung

Die Abbildung 62 zeigt, dass das Hinzufügen eines Datenelements zu einer Liste kein Aktualisieren des Datagrids verursacht. Der Grund dafür liegt darin, dass der Setter der Eigenschaft bei Hinzufügen eines Datensatzes zu einer Liste nicht aufgerufen wird und es so nicht zum Auslösen des PropertyChanged Events kommt, das zum Aktualisieren des Datagrids führen würde.

Die Lösung des Problems ist die Nutzung des Typs ObservableCollection. Dieser Typ stellt nicht nur eine dynamische Datenauflistung dar, sondern reagiert auch mit dem Auslösen des PropertyChanged Events, wenn Elemente hinzugefügt oder gelöscht werden.

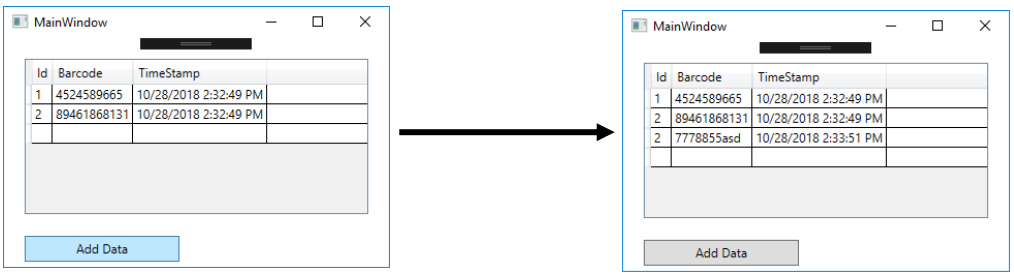


Abbildung 63: Versuch 2 Early Prototyping Benutzeroberfläche, Quelle: Eigene Darstellung

Der Versuch zeigt, dass die Verwendung von Listen in dieser Anwendung nicht den gewünschten Erfolg bringt. Erst der Einsatz des ObservableCollection Typs ermöglicht das automatische Aktualisieren des Steuerelements beim Verändern der Daten innerhalb der Collection.

8 AUTOMATISIERTE LEISTUNGSTESTS

Neben dem Funktionsnachweis von Logistikanlagen ist auch der Nachweis der verkauften Leistung Teil dieser Arbeit. Dieser Nachweis soll durch die Definition der Tests, die automatisierte Vorbereitung und die anschließende Auswertung der gemessenen Daten erbracht werden. Die Leistung der Regalbediengeräte im Hochregallager wird durch die Messung der FEM-Spielzeiten berechnet. Im Bereich der Fördertechnik soll der Durchsatz an definierten Eckpunkten der Anlage gemessen und so auf die Gesamtleistung der Anlage zurückgerechnet werden können.

Ziel dieses Kapitels ist es, die Anforderungen an die automatisierten Leistungstests zu definieren und anschließend davon ein Lösungskonzept abzuleiten. Dabei wird zwischen dem allgemeinen Teil und den Leistungstests für Regalbediengeräte und für Fördersysteme unterschieden.

8.1 Anforderungen

Die Anforderungen an die automatisierten Leistungstests wurden teilweise schon in Kapitel 2.4.2 definiert. Dieses Unterkapitel geht nun genauer auf die technischen Anforderungen der Leistungstests ein.

- Die Tests sollen getrennt von der Anwendungssoftware erstellt werden, da diese noch nicht fertig entwickelt ist und die Leistungstests möglichst unabhängig von etwaigen Veränderungen bleiben sollen.
- Der Zugriff auf die Messdaten darf nicht über die Datenbank erfolgen, da die Messdaten einfach an andere weitergegeben werden können sollen und die Grundlage für die nachgewiesene Leistung sind.
- Die Leistungstests sollen über die Benutzeroberfläche angelegt, vorbereitet und zur Laufzeit mitverfolgt werden können.
- Die Leistungstests des Regalbediengerätes sollen mit bereits abgeschlossenen Tests verglichen werden können.
- Die Leistungstests sollen unabhängig von den zu emulierenden Warehouse-Management-Systemen funktionieren.
- Die Kommunikationszeit zwischen Steuerungsebene und Emulator soll bei der Messung der Leistung berücksichtigt werden.

8.2 Lösungsweg

Um die Leistungstests getrennt von der Anwendungssoftware entwickeln zu können, werden sie der Architektur der Benutzeroberfläche zugeordnet. Die Messung der Leistung soll von einer eigenen Klasse durchgeführt werden, die in der ViewModel Klasse des Leistungstestbildes instanziiert werden kann.

Messdaten

Um die Leistung der Regalbediengeräte oder des Fördersystems messen zu können, werden Daten benötigt. Diese Daten können nur von der Anwendungssoftware erstellt werden, da nur dieser Teil des Emulators eine Verbindung zur Steuerungsebene hat. Um Zugriff auf die in der Anwendungssoftware erhobenen Daten zu erhalten, ohne diese in der Datenbank ablegen zu müssen, muss ein Ereignislogfile

erstellt werden, in dem alle Ereignisse der Steuerungsebene abgelegt werden können. Dieses Logfile soll nicht nur während des Leistungstests geführt werden, sondern während der gesamten Inbetriebnahme bei der Fehlersuche unterstützen.

Datenformat

Da sich die Ereignisse des Regalbediengerätes und der Fördertechnik stark unterscheiden, müssen die zu speichernden Informationen nicht zwingend identisch sein. Das Dateiformat der Logfiles soll bei beiden Systemen gleich sein.

Das gewählte Dateiformat des Ereignisprotokolls soll folgende Anforderungen erfüllen:

- Möglichst geringer Overhead, da die Dateigröße trotz der Anzahl an Daten möglichst gering bleiben soll
- Einfaches Speichern und Lesen von Daten
- Einfache Lesbarkeit der Daten für den Benutzer, um bei der Fehlersuche schnell die gewünschten Einträge zu finden

Die genannten Anforderungen können von einer Comma-Separated-Value (CSV) Datei am besten erfüllt werden. Der Overhead der CSV-Datei ist kleiner als bei JSON und die darin enthaltenen Informationen können von der Anwendung und dem Benutzer einfacher gelesen werden als bei einer Text Datei.

Die Kommunikation der Anwendung mit dem Leistungstest über das Ereignisprotokoll sowie die Einordnung in die Architektur des Emulators sind in der Abbildung 64 grafisch dargestellt.

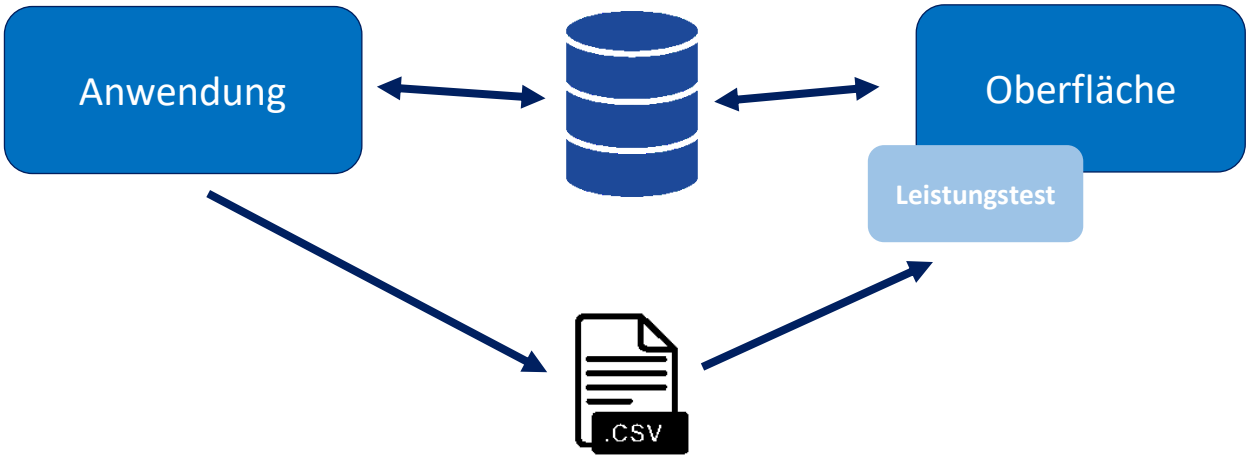


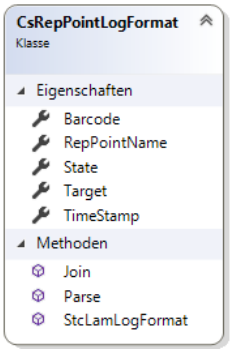
Abbildung 64: Einordnung des Leistungstests in die Architektur des Emulators, Quelle: Eigene Darstellung

Um die Ereignisse innerhalb des Logfiles den Meldepunkten zuordenbar zu machen, soll für jeden Punkt eine CSV-Datei erstellt werden. Der Name der Datei soll sich dabei aus der Identifikationsnummer des Meldepunktes und dessen Namen zusammensetzen. Das Schreiben und Lesen des Logfiles soll von einer eigenen Klasse übernommen werden, die in der Klasse des Basismeldepunktes instanziiert wird.

Überschreitet die Größe des Logfiles ein Megabyte (MB), soll die Klasse das File selbstständig in einen Archivordner kopieren und ein neues File erstellen.

Für die zu speichernden Informationen wird ein eigenes Format für die RBG-Einträge und für die Fördertechnik-Einträge erstellt. Für beide Formate soll eine Klasse entwickelt werden, die die zu speichernden Daten selbständig in das CSV Format bringt und beim Lesen wieder zurückwandelt.

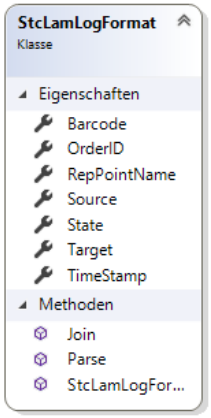
Wie In Abbildung 65 ersichtlich besteht ein Log-Eintrag der Fördertechnik aus dem Zeitstempel, dem Namen des Meldepunktes, seinem Status sowie dem Ziel und Barcode des Ladehilfsmittels, das über den Meldepunkt fährt.



Timestamp	RepPointName	State	Target	Barcode
14.10.2018 20:21	IP01	Ready	MP11	23036
14.10.2018 20:21	IP01	Ready	MP11	27409
14.10.2018 20:21	IP01	Ready	MP11	77408
14.10.2018 20:21	IP01	Ready	MP11	48053
14.10.2018 20:21	IP01	Ready	MP11	83512
14.10.2018 20:21	IP01	Ready	MP11	39634
14.10.2018 20:21	IP01	Ready	MP11	18701
14.10.2018 20:21	IP01	Ready	MP11	97486
14.10.2018 20:21	IP01	Ready	MP11	78002
14.10.2018 20:21	IP01	Ready	MP11	30919
14.10.2018 20:21	IP01	Ready	MP11	62904
14.10.2018 20:21	IP01	Ready	MP11	24237

Abbildung 65: Datenformat Ereignislogfile Fördertechnik, Quelle: Eigene Darstellung

Der Log-Eintrag des Lastaufnahmemittels beinhaltet wie in Abbildung 66 dargestellt ebenfalls den Zeitstempel des Ereignisses und den Namen des Meldepunktes. Im Unterschied zu den Ereignissen der Fördertechnik bezieht sich jedoch der eingetragene Status nicht auf den Meldepunkt, sondern auf den Auftrag des LAM, wodurch die einzelnen Phasen des Auftrages sichtbar werden. Zusätzlich beinhaltet das Log den Barcode des Ladehilfsmittels, dessen Quelle und Ziel sowie die Nummer des durchgeführten Auftrages.



TimeStamp	RepPointName	OrderID	State	Barcode	Source	Target
13.10.2018 12:31	STC11-1	21	New	1A2341343B4zzz435	PS1	STC11-1
13.10.2018 12:31	STC11-1	21	Drive	1A2341343B4zzz435	PS1	STC11-1
13.10.2018 12:31	STC11-1	21	Cycle	1A2341343B4zzz435	PS1	STC11-1
13.10.2018 12:31	STC11-1	21	Ready	1A2341343B4zzz435	PS1	STC11-1
13.10.2018 12:31	STC11-1	21	New	1A2341343B4zzz435	STC11-1	G1-1-4
13.10.2018 12:31	STC11-1	21	Drive	1A2341343B4zzz435	STC11-1	G1-1-4
13.10.2018 12:31	STC11-1	21	Cycle	1A2341343B4zzz435	STC11-1	G1-1-4
13.10.2018 12:31	STC11-1	21	Ready	1A2341343B4zzz435	STC11-1	G1-1-4

Abbildung 66: Datenformat Ereignislogfile Lastaufnahmemittel, Quelle: Eigene Darstellung

Datenverfälschung durch Kommunikationszeit

Für das Berechnen der Leistung ist es erforderlich, dass die Zeitstempel der im Ereignisprotokoll eingetragenen Daten die Aktionen der Steuerungstechnik zeitlich exakt widerspiegeln. Durch die benötigte

Kommunikationszeit zwischen Steuerungsebene und dem Emulator entsteht jedoch eine Latenzzeit, die das Ergebnis des Leistungstests verfälscht. Diese Zeit kann entweder durch einen konstanten Offset oder durch die Messung der realen Kommunikationszeit entfernt werden. Da sowohl das Kommunikationsprotokoll des SAP-EWM als auch das des Jungheinrich-WMS eine solche Messung einfach zulässt, soll diese Variante bevorzugt werden.

Beide Warehouse-Management-Systeme verwenden ein TCP-Kommunikationsprotokoll mit Telegrammquittung. Wie in Abbildung 67 dargestellt muss dabei jedes gesendete Telegramm vor der Verarbeitung mit einer Quittung bestätigt werden. Wird die Zeit zwischen dem gesendeten Telegramm und der erhaltenen Quittung gemessen und halbiert, erhält man die mittlere Kommunikationszeit eines Telegrammes.

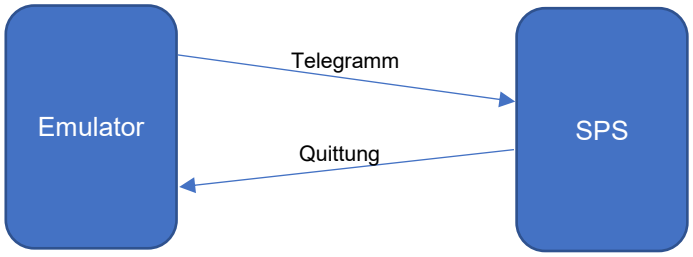


Abbildung 67: Telegrammquittierung eines vom Emulator gesendeten Telegrammes, Quelle: Eigene Darstellung

Die gemessene Kommunikationszeit soll bei der Protokollierung der Ereignisse im Logfile berücksichtigt werden.

8.3 Lösungsweg Regalbediengerät

Der Leistungstest des Regalbediengerätes besteht aus vier Phasen: Der Definition, der Vorbereitung, der Messung und dem Vergleich. In jeder der vier Phasen werden vom Emulator unterschiedliche Funktionen ausgeführt.

8.3.1 Definition

In dieser Phase des Leistungstest werden die Fahraufträge des Tests über die Bedienoberfläche des Emulators angelegt. Durch Klicken des „Neu“-Buttons im rechten oberen Teil des Fensters öffnet sich der in Abbildung 68 gezeigte Dialog für das Erstellen der Aufträge. Im ersten Dropdownfeld kann das zu testende Regalbediengerät ausgewählt, über die Felder darunter dann die Quelle sowie das Ziel des Auftrages angegeben werden. Über den „Plus“-Button auf der rechten Seite wird der Auftrag dem Test hinzugefügt.

Beim Aufrufen des Dialogs werden automatisch alle Namen der RBG-Steuerungen in das erste Dropdownfeld geladen. Nach Auswahl der RBG-Steuerung über die Combobox werden auch alle weiteren Felder des Dialoges befüllt. Über das Feld „Typ“ im Abschnitt „Source“ kann ausgewählt werden, ob von einer Einlagerungsstation oder einer Position im Lager aufgenommen werden soll. Nach Auswahl einer Einlagerungsstation werden alle weiteren Felder, die das Lagerfach spezifizieren könnten, gesperrt. Die Funktion des Target-Abschnittes ist fast ident mit der des Source-Abschnittes, aber mit dem Unterschied, dass nur Auslagerungsstationen oder Lagerfächer gewählt werden können.

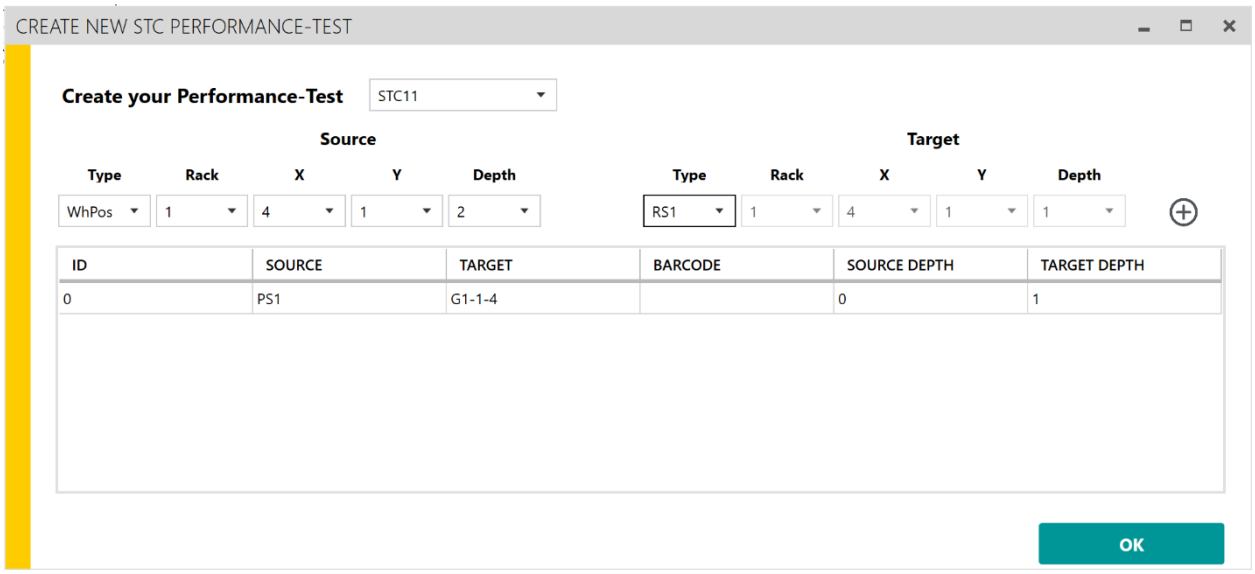


Abbildung 68: Dialog zum Erstellen der Fahraufträge des Leistungstests, Quelle: Eigene Darstellung

Sowohl die Identifikationsnummer als auch der Barcode des Auftrages bleiben in dieser Phase noch leer. Erst in der Vorbereitungsphase wird der Auftrag in der Datenbank angelegt, wodurch er auch eine eindeutige Nummer erhält. Der Barcode wird erst während der Messung eingetragen, da vorab noch nicht festgestellt werden kann, welches Ladehilfsmittel später im Test von der Einlagerungsstation aufgenommen wird.

8.3.2 Vorbereitung

Bei Start des Leistungstests kann nicht davon ausgegangen werden, dass alle definierten Aufträge vom Regalbediengerät gefahren werden können. Speziell wenn Leistungstests wiederholt werden, kann es vorkommen, dass auf den Aufnahmepositionen der Aufträge kein Ladehilfsmittel steht oder die Abgabepositionen bereits belegt sind. Um solchen Situationen vorzubeugen, wird der Leistungstest vor dem Start vorbereitet. Während dieser Vorbereitungsphase werden Ladehilfsmittel innerhalb der Gasse so umgelagert, dass alle Aufnahmepositionen belegt und alle Abgabepositionen frei sind. Zusätzlich wird überprüft, ob die Positionen wie in Abbildung 69 nicht von anderen LHM blockiert werden und ob durch die Abgabe der LHM keine Lücken im Kanal entstehen.

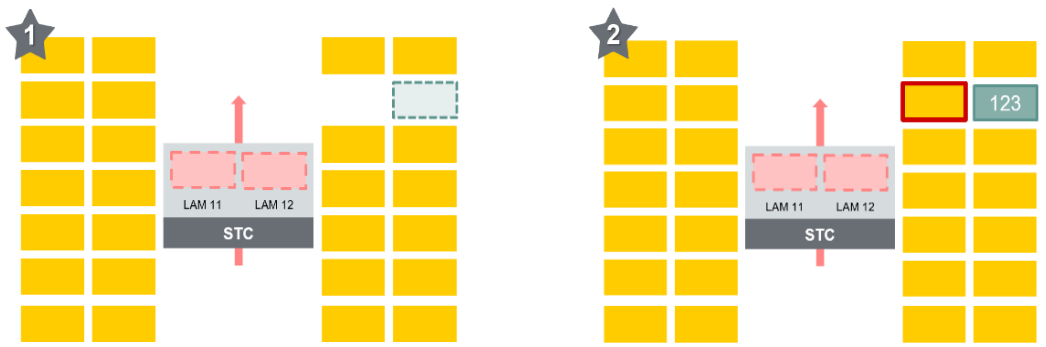


Abbildung 69: Prüfen der Aufnahmeposition bei der Leistungstestvorbereitung, Quelle: Eigene Darstellung

Fällt bei der Prüfung der Aufnahme- und Abgabepositionen auf, dass ein Auftrag nicht durchgeführt werden kann, wird ein neuer Umlagerauftrag in der Datenbank Tabelle tblVehicleOrder angelegt. Dieser Vorgang wird solange durchgeführt, bis alle definierten Aufträge vom RBG umgesetzt werden können.

Um die laufende Vorbereitungsphase dem Benutzer zu visualisieren, wird im „Start"-Button des Leistungstests ein Progress Ring angezeigt. Erst nachdem alle Vorbereitungen abgeschlossen sind, stoppt der Ring und der Button wird zur Bedienung freigegeben.

Abbildung 70: Visualisierung der Vorbereitungsphase anhand des Start Button, Quelle: Eigene Darstelung

Neben dem Überprüfen der Aufträge auf ihre Machbarkeit werden alle bereits vom System erstellten Aufträge für etwaige Ein- und Auslagerungen des Normalbetriebs in den Status „Hold" gebracht. Der Status „Hold" wird extra für die Durchführung von Leistungstests entwickelt und blockiert alle bereits angelegten Aufträge bis der Leistungstest abgeschlossen ist.

8.3.3 Messung

Bei der Leistungsmessung von Regalbediengeräten müssen alle definierten Aufträge in einer bestimmten Zeit abgearbeitet werden. Das Endergebnis setzt sich aus der Summe der für die Einzelaufträge benötigten Zeit zusammen. Das reine Bilden der Differenz aus End- und Startzeitpunkt würde nicht das gewünschte Ergebnis bringen, da dabei weder die Kommunikationszeit noch die Rechenzeit des Emulators berücksichtigt werden würde. Alle Zwischenzeiten, die nicht direkt für den Auftragsablauf relevant sind, werden nur zur besseren Nachvollziehbarkeit des Testergebnisses verwendet und um den Test mit anderen vergleichbar zu machen. Je mehr Zwischenzeitmessungen des Auftrages protokolliert werden, desto genauer kann das Ergebnis interpretiert werden und desto mehr mögliche Schwachstellen können gefunden werden. Das Kommunikationsprotokoll des Jungheinrich-WMS ermöglicht es, sechs Zwischenzeiten für einen Auftrag zu protokollieren, wobei das von SAP-EWM keine einzige zulässt. Der Unterschied bei der Ereignisprotokollierung der zwei Systeme zeigt sich in Abbildung 71 und Abbildung 72.

Timestamp	RepPointName	OrderID	State	Barcode	Source	Target
13.10.2018 12:31	STC11-1	21	New	1A2341343B	PS1	STC11-1
13.10.2018 12:31	STC11-1	21	Drive	1A2341343B	PS1	STC11-1
13.10.2018 12:31	STC11-1	21	Cycle	1A2341343B	PS1	STC11-1
13.10.2018 12:31	STC11-1	21	Ready	1A2341343B	PS1	STC11-1
13.10.2018 12:31	STC11-1	21	New	1A2341343B	STC11-1	G1-1-4
13.10.2018 12:31	STC11-1	21	Drive	1A2341343B	STC11-1	G1-1-4
13.10.2018 12:31	STC11-1	21	Cycle	1A2341343B	STC11-1	G1-1-4
13.10.2018 12:31	STC11-1	21	Ready	1A2341343B	STC11-1	G1-1-4

Abbildung 71: Auszug Ereignislogfile RBG Jungheinrich-WMS, Quelle: Eigene Darstellung

Timestamp	RepPointName	OrderID	State	Barcode	Source	Target
13.10.2018 12:31	STC11-1	21	New	1A2341343B	PS1	STC11-1
13.10.2018 12:31	STC11-1	21	Ready	1A2341343B	STC11-1	G1-1-4

Abbildung 72: Auszug Ereignislogfile RBG SAP-EWM, Quelle: Eigene Darstellung

Aus dem Ereignisprotokoll des Lastaufnahmemittel-Meldepunktes in Abbildung 71 können insgesamt neun Einzelzeiten berechnet werden:

1. Auftragsprüfung des Aufnahmeauftrages
2. Fahrt zur Aufnahmeposition
3. Aufnahme
4. Gesamte Aufnahmedauer
5. Auftragsprüfung des Abgabeauftrages
6. Fahrt zur Abgabeposition
7. Abgabe
8. Gesamte Abgabedauer
9. Gesamte Auftragsdauer

Da das Ereignislogfile nicht nur die Ereignisse des Leistungstests beinhaltet, müssen die einzelnen Einträge den Aufträgen des Tests zugeordnet werden. Dies kann über die Identifikationsnummer (ID) des Auftrages im Logfile erreicht werden. Jeder Auftrag des Leistungstests hat in der Datenbank eine eindeutige ID, die den Einträgen des Logfiles zugeordnet werden kann. Auf diese Weise können die Start- und Endzeitpunkte sowie alle Zwischenzeiten in der Tabelle des Leistungstestbildes auf der Benutzeroberfläche angezeigt werden.

Die Anzeige der Messwerte erfolgt neben dem Eintrag in der Tabelle auch grafisch mittels Liniendiagram. Dieses Diagramm passt sich während des Tests laufend an die Messergebnisse an und erlaubt dem Benutzer so, den aktuellen Status des Tests zu erfassen. Neben den aktuellen Messwerten wird wie in Abbildung 73 auch das Zielergebnis des Leistungstests als Gerade in dem Diagramm eingezeichnet. Diese Linie soll den ständigen Vergleich mit dem Leistungsziel des Tests ermöglichen.

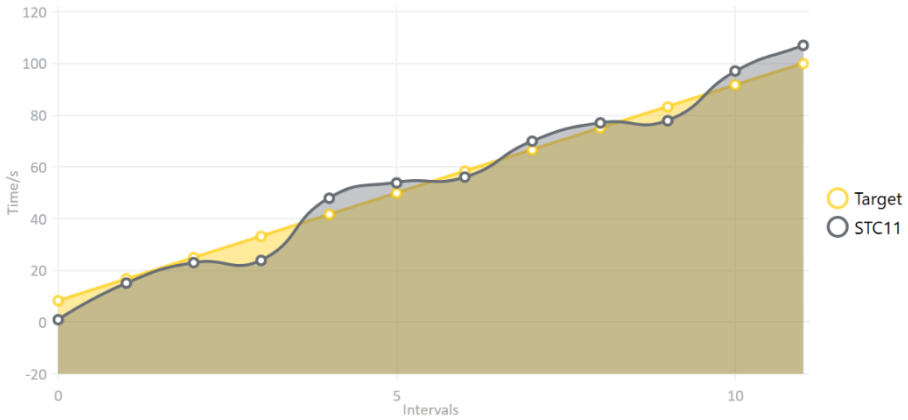


Abbildung 73: Liniendiagram Leistungstest RBG, Quelle: Eigene Darstellung

Neben dem Liniendiagram zur grafischen Darstellung aller Messwerte zeigt ein Gauge-Diagramm die aktuell erreichte Leistung des Regalbediengerätes in Prozent an. Das Leistungsziel wird dabei als 100% der Leistung angenommen, wodurch ein erfolgreicher Leistungstest erst bei 100% beginnt und weit über diesen Wert gehen kann. Die Abbildung 74 zeigt das Diagramm bei einem aktuellen Wert von 93,5%.

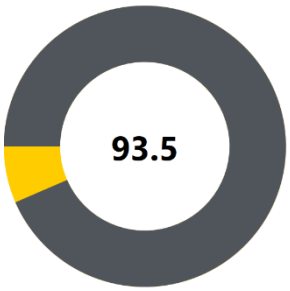


Abbildung 74: Gauge-Diagramm Leistungstest RBG, Quelle: Eigene Darstellung

Nach dem Beenden des Leistungstests können die Messwerte und die gefahrenen Aufträge gespeichert werden. Dazu werden die im Datagrid angezeigten Daten sowie der Leistungszielwert in eine binäre Datei serialisiert. Der Datei-Typ der serialisierten Datei ist SPT0 und steht für Stacker Crane Performance Test Version 0. Auf Wunsch kann diese Datei auf der Benutzeroberfläche über den Button „Öffnen“ auch wieder deserialisiert werden. Nach dem Öffnen der Datei kann der gespeicherte Test wiederholt gestartet oder mit anderen gespeicherten Tests verglichen werden. Die Abbildung 75 zeigt die Oberfläche des Leistungstests mit dem Datagrid und den zwei Diagrammen.

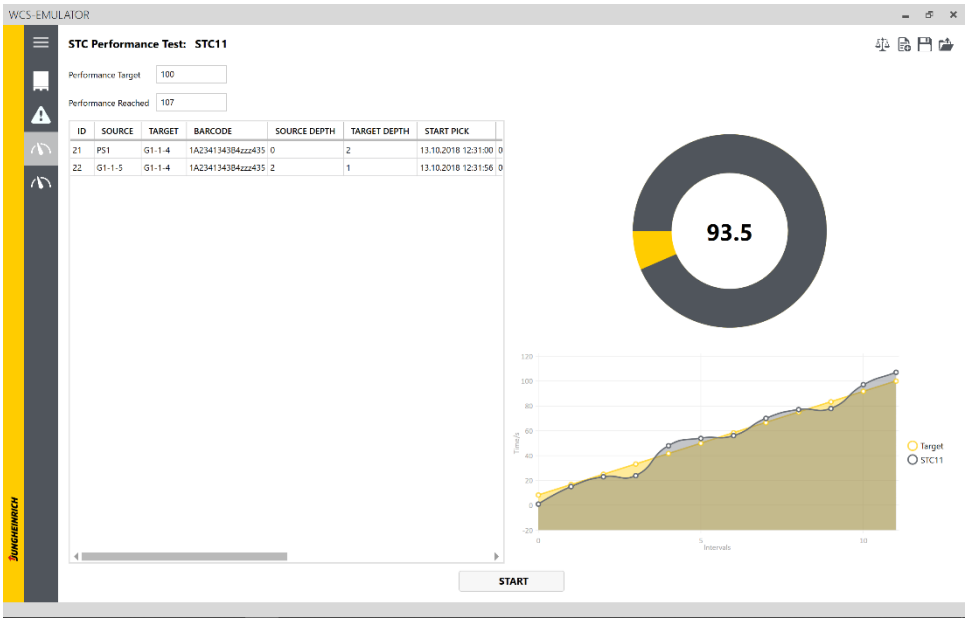


Abbildung 75: Leistungstest RBG Bedienoberfläche, Quelle: Eigene Darstellung

8.3.4 Vergleich

Wie schon im vorangegangenen Kapitel erwähnt können Testergebnisse verglichen werden. Dieser Vergleich erfolgt dabei rein grafisch über die zwei Diagramme der Bedienoberfläche.

Um den Vergleich zu beginnen, müssen die Daten eines bereits fertiggestellten Leistungstest auf der Oberfläche angezeigt werden. Das kann durch Starten eines neuen Tests oder durch Öffnen eines bestehenden erreicht werden. Als Vergleich kann anschließend ein weiterer gespeicherter Datensatz ausgewählt werden. Der Vergleichsdatensatz wird dann deserialisiert und die Messwerte des Tests werden durch eine weitere Linie im Liniendiagramm visualisiert. Zusätzlich zeigt ein zweites Gauge-Diagramm die Leistung des Vergleichstests in Bezug auf den Zielwert in Prozent an.

Die eben beschriebene Vergleichsoption kann der Abbildung 76 entnommen werden.

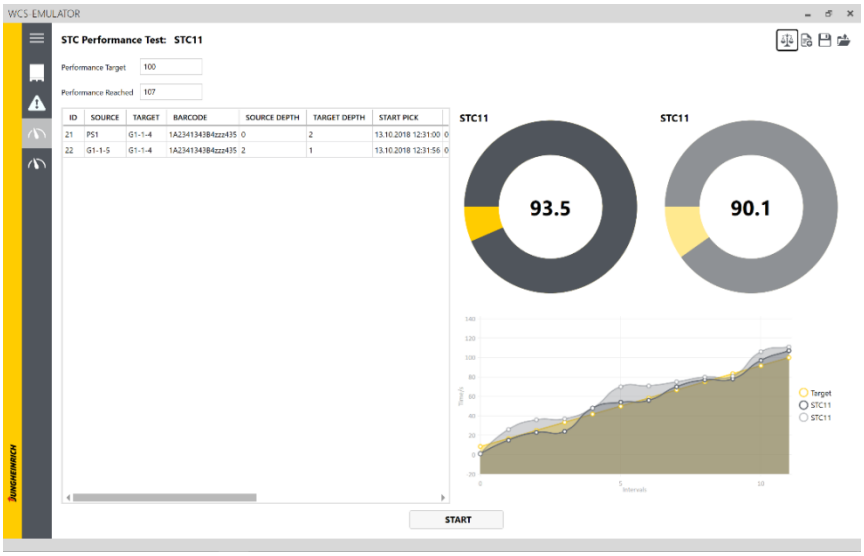


Abbildung 76: Bedienoberfläche Leistungstestvergleich RBG, Quelle: Eigene Darstellung

8.4 Lösungsweg Fördertechnik

Der Leistungstest der Fördertechnik besteht anders als der der RBG nur aus drei Phasen: Der Definition, der Vorbereitung und der Messung. Das Vergleichen von Leistungstests wird auf der Fördertechnik nicht benötigt, da im Gegensatz zum Regalbediengerät kein Rückschluss auf einzelne Teilschritte getroffen werden kann.

8.4.1 Definition

Bei der Definition kann der Benutzer die zu messenden Meldepunkte und deren Zielleistung sowie jene Meldepunkte definieren, die zum Aufstauen von Ladehilfsmitteln verwendet werden sollen. Über die Schaltfläche „Neu“ auf der Benutzeroberfläche kann der Dialog aufgerufen werden. Auf der linken Seite des Dialoges, der in Abbildung 77 dargestellt wird, können alle Meldepunkte angegeben werden, die bei der Vorbereitung des Leistungstest mitwirken sollen. Auf der rechten Seite können jene Meldepunkte definiert werden, die für die Messung der Leistung relevant sind und deren Leistung in Ladehilfsmittel pro Stunde angegeben wird.

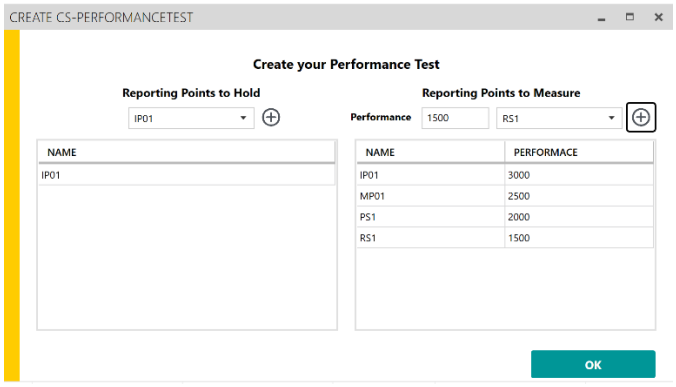


Abbildung 77: Dialog zum Erstellen des Fördertechnik Leistungstests, Quelle: Eigene Darstellung

Die Dropdownfelder des Dialogs werden wie schon beim Dialog des RBG beim Öffnen mit den Meldepunkten der Datenbank befüllt. Nach dem Bestätigen durch die „OK“-Schaltfläche schließt sich der Dialog und die eingegeben Daten werden im Hintergrund der Anwendung gespeichert.

8.4.2 Vorbereitung

Das Ziel dieses Teils ist – im Unterschied zur Vorbereitungsphase des RBG Tests – nicht, dass der Test umgesetzt werden kann, sondern dass die zu messende Leistung auch tatsächlich auf die Anlage gebracht werden kann. Um das zu erreichen, werden alle zuvor als „Reporting Points to Hold“ deklarierten Meldepunkte in den Status „Hold“ gebracht. Dieser Status bewirkt, dass die Meldepunkte über die maximale Kapazität hinaus befüllt werden und der Meldepunkt keine weiteren Transportaufträge an die Steuerungsebene sendet. Wie in Abbildung 78 ersichtlich kommt es dadurch zum Stau vor dem Meldepunkt, in diesem Beispiel vor dem Identifikationspunkt 1 (IP01).

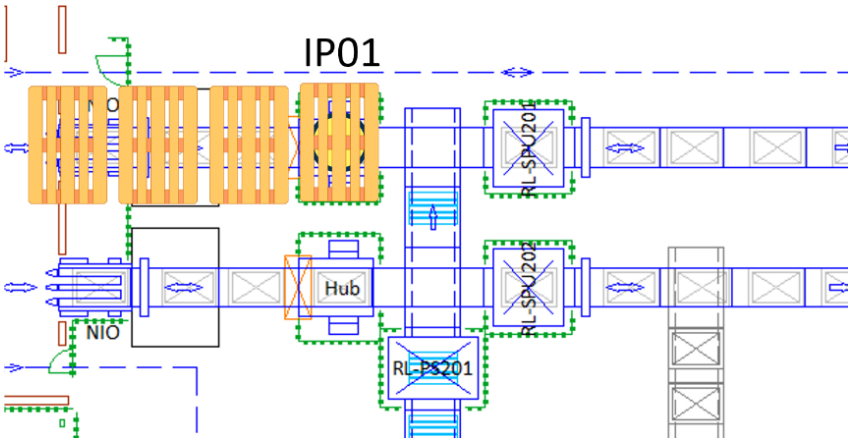


Abbildung 78: Vorbereitung Leistungstest Fördertechnik, Quelle: Eigene Darstellung

Durch diesen Vorgang kann die Anlage vor dem Start des Leistungstest bis zum Maximum befüllt werden, was eine bestmögliche Auslastung der Fördertechnik während des Tests ermöglicht.

8.4.3 Messung

Nach Beginn des Tests werden alle Meldepunkte gleichzeitig vom Status „Hold“ in den Status „Ready“ gebracht. In diesem Status dürfen wieder Transportaufträge gesendet werden, wodurch sich der während der Vorbereitung aufgebaute Stau langsam auflöst.

Bei der Leistungsmessung der einzelnen Meldepunkte zählt die Anzahl der Ladehilfsmittel, die den Meldepunkt innerhalb einer vordefinierten Zeit passieren. Die Protokollierung der einzelnen Zeitstempel der LHM dient wieder nur der Nachvollziehbarkeit der Messwerte. Alle Ereignisse der gemessenen Meldepunkte werden wie in Abbildung 79 in der Tabelle der Benutzeroberfläche angezeigt.

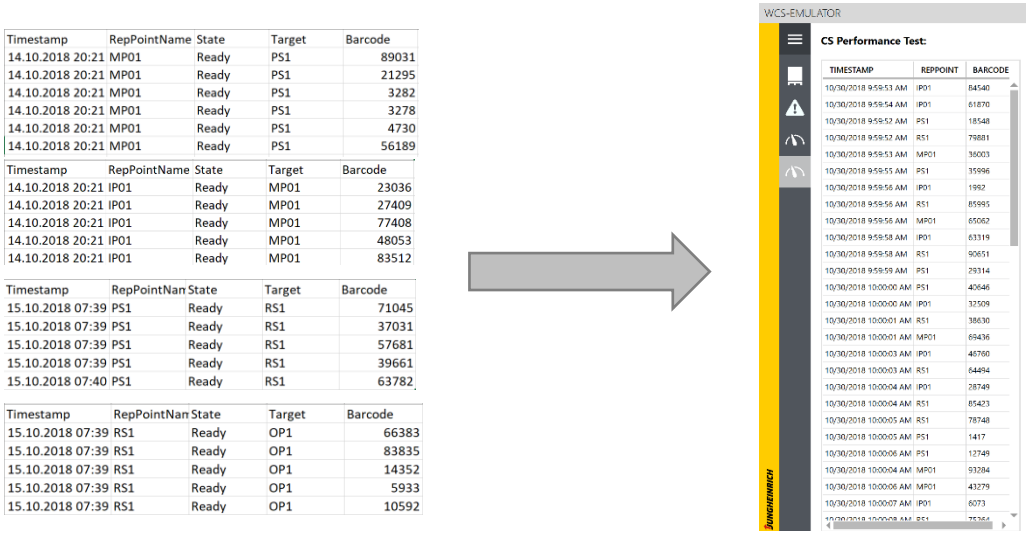


Abbildung 79: Anzeigen der Daten des Logfiles in der Benutzeroberfläche, Quelle: Eigene Darstellung

Für die Berechnung der aktuellen Leistung des Meldepunktes wird die Zeit zwischen zwei Ladehilfsmitteln gemessen und durch 60 Minuten dividiert. Es ergibt sich daraus die aktuelle Leistung in LHM pro Stunde. Für die Anzeige der aktuellen Leistung und der Gesamtleistung jedes einzelnen ausgewählten Meldepunktes werden ähnlich wie beim Regalbediengerät ein Linien- und ein Gauge-Diagramm verwendet. Die gewünschten Daten können über die Auswahl des Meldepunktes mittels Dropdownfeld in den Diagrammen angezeigt werden. Zur Auswahl stehen dabei nur Meldepunkte, die aktiv während des Tests gemessen werden. Die Abbildung 80 zeigt die dynamische Anzeige der Leistungsdaten eines Meldepunktes. Das Gauge-Diagramm visualisiert die Gesamtleistung des Punktes in Prozent bezogen auf die Zielleistung, die während der Definitionsphase vorab festgelegt wurde. Im Liniendiagramm werden die letzten zehn Messwerte sowie die Zielleistung als Gerade angezeigt.

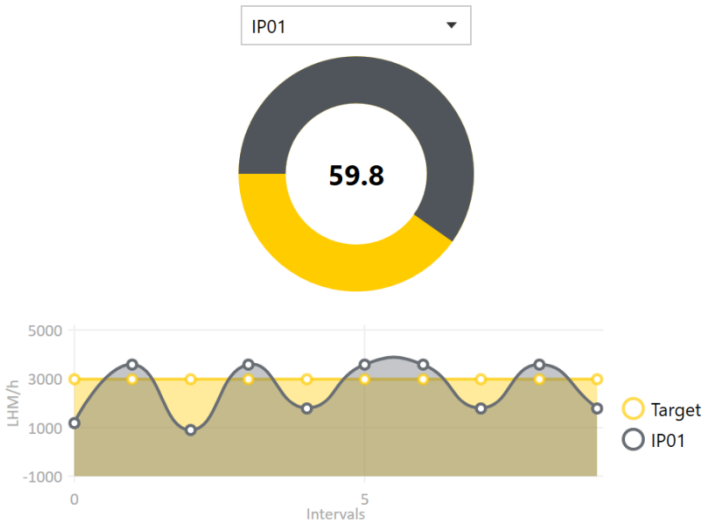


Abbildung 80: Dynamische Anzeige der Leistung des Meldepunktes, Quelle: Eigene Darstellung

Auch dieser Leistungstest kann über die Benutzeroberfläche gespeichert und geöffnet werden. Gesichert werden dabei die angegebenen Daten der Definitionsphase sowie alle Messwerte. In der Abbildung 81 wird die gesamte Seite des Leistungstests der Fördertechnik dargestellt.

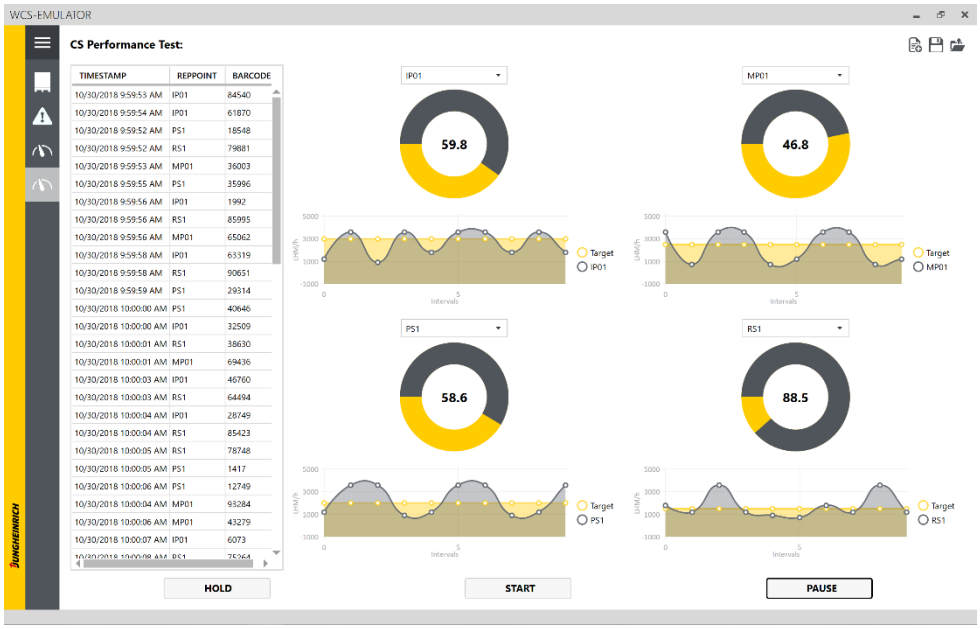


Abbildung 81: Leistungstest Fördertechnik Bedienoberfläche, Quelle: Eigene Darstellung

8.5 Funktionstests

Um die Funktion der automatisierten Leistungstests nachweisen zu können, sollen zwei Testanwendungen entwickelt werden, die es ermöglichen, die Leistungstests ohne den Anwendungsteil und die Anlage zu prüfen.

8.5.1 Leistungstest Regalbediengerät

Die Überprüfung des RBG-Leistungstests soll die Vorbereitung und die Messung testen können. In der Vorbereitungsphase sollen die vom Leistungstest erstellten Umlageraufträge von der Testanwendung in den Zustand „Done“ gebracht werden und die Ladehilfsmittel vom Quelllagerfach auf das Ziellagerfach gebucht werden. Während der Messphase des Leistungstestes ist die Aufgabe der Testanwendung, Logeinträge für alle definierten Aufträge des Tests im Ereignisprotokoll zu erstellen. Die Abstände zwischen den Einträgen sollen zufällig zwischen einer Sekunde und fünf Sekunden sein. Die Vorbereitung der Datenbank und die Evaluierung der Testergebnisse erfolgen dabei händisch. Der gesamte Ablauf der Überprüfung des Leistungstest kann dem Ablaufdiagramm in Abbildung 82 entnommen werden. Alle gelben Aktionen sind dabei händisch zu bearbeiten, alle grünen werden von der Leistungstest-Anwendung ausgeführt und alle blauen von der Testanwendung. Am Ende des Tests soll das Ergebnis händisch evaluiert werden, wobei die angezeigten Werte mit dem Ereignisprotokoll verglichen werden. Ist das Testergebnis korrekt, gilt der Test als erfolgreich. Stimmt das Testergebnis nicht mit den Messungen des Ereignisprotokolls überein oder konnte der Test nicht korrekt vorbereitet werden, sollen die Fehler in der Anwendung behoben und anschließend der Test neu gestartet werden.

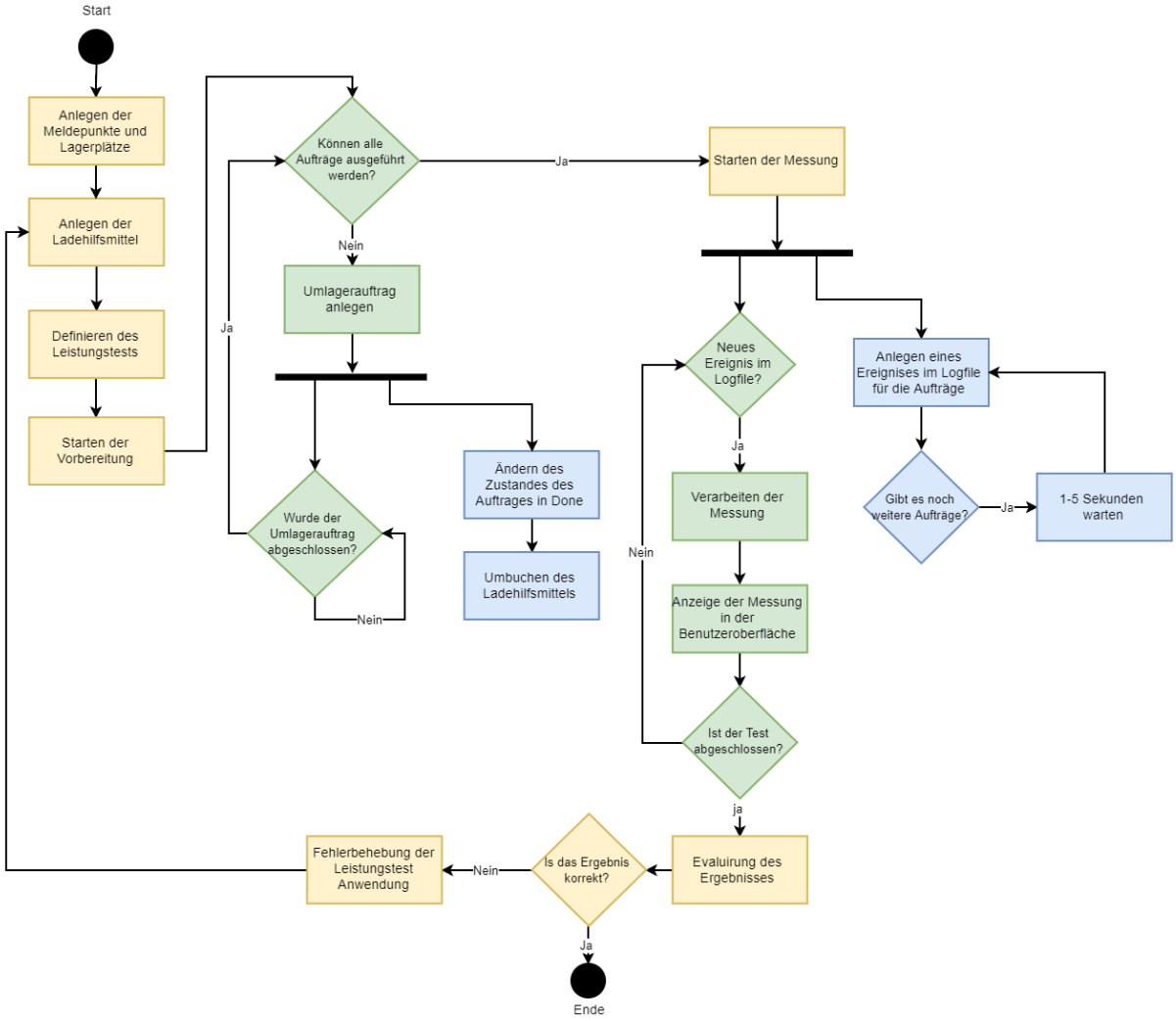


Abbildung 82: Ablauf der Überprüfung der RBG Leistungstest Anwendung, Quelle: Eigene Darstellung

Testergebnis:

Während des Tests ist aufgefallen, dass es zu Problemen kommen kann, wenn der Thread der Testanwendung und jener der Leistungstest-Anwendung gleichzeitig auf das Ereignisprotokoll zugreifen. Der Fehler kann behoben werden, indem das Öffnen der Datei bei einem Fehler nach 10ms erneut versucht wird.

8.5.2 Leistungstest Fördertechnik

Auch die Vorbereitungs- und Messfunktion des Fördertechnik Leistungstests muss überprüft werden. Die Testanwendung soll zuerst untersuchen, ob nach dem Start der Vorbereitung alle definierten Meldepunkt in den Zustand „Hold“ gebracht werden. Nach Beginn des Leistungstests soll die Anwendung in zufälligen Zeitabständen Ereignisse in den Logfiles der zu messenden Meldepunkte ablegen. Nach dem Stoppen der Anwendung soll das Ergebnis des Tests händisch mittels Einträge der Logfiles evaluiert werden. Ist das Ergebnis des Leistungstests korrekt, gilt der Test als erfolgreich. Konnten Fehler während des Ablaufs festgestellt werden, sollen diese behoben und anschließend der Test wiederholt werden. Der Ablauf der Funktionsüberprüfung kann dem Ablaufdiagram in Abbildung 83 entnommen werden. Alle gelb markierten

Aktionen sind dabei wieder händisch durchzuführen, grüne sollen von der Leistungstest-Anwendung und blaue von der Testanwendung bearbeitet werden.

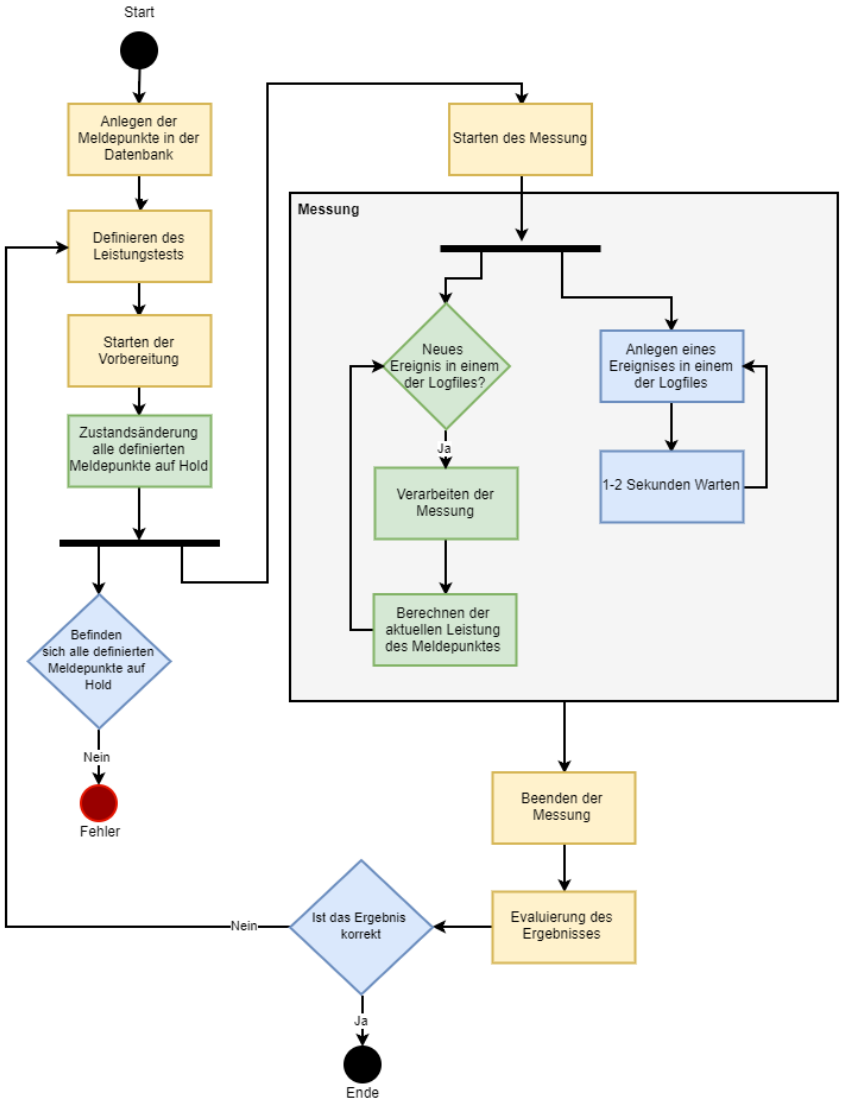


Abbildung 83: Ablauf der Überprüfung der Fördertechnik Leistungstest Anwendung, Quelle: Eigene Darstellung

Testergebnis:

Während des Tests wurde deutlich, dass maximal die letzten zehn Messwerte im Liniendiagramm der Benutzeroberfläche angezeigt werden dürfen. Mehr Messwerte machen das Diagramm unübersichtlich und schwer ablesbar.

9 REPLAY-FUNKTION

Den letzten Teil dieser Arbeit bildet die Konzeptionierung einer Replay-Funktion. Diese soll den Inbetriebnehmer beim Testen der Automatikfunktion unterstützen und die Fehlersuche vereinfachen. Ziel dieses Kapitels ist es, die Anforderungen an die Replay-Funktion zu definieren und anschließend ein Lösungskonzept abzuleiten.

9.1 Anforderungen

Die Anforderungen an die Replay-Funktion wurden zum Teil schon in Kapitel 2.4.3 definiert. In diesem Kapitel soll nun genauer auf die technischen Details der Funktion eingegangen werden:

- Die Funktion soll unabhängig vom emulierenden Warehouse-Management-System und der Kommunikationsschnittstelle sein.
- Möglichst eindeutige Zuordnung der Funktion zur Anwendung oder zur Oberfläche des Emulators.
- Pro Meldepunkt sollen bis zu zehn Bewegungen wiederholt werden können.
- Über eine „Pause“ Schaltfläche auf der Benutzeroberfläche sollen bestimmte oder alle Meldepunkte der Anlage angehalten werden können.
- Über die Bedienoberfläche soll ein Zeitpunkt gewählt werden können, auf den die ausgewählten Meldepunkte zurückgesetzt werden.
- Nach dem Start der Funktion sollen alle Ladehilfsmittel zeitlich gleich versetzt wie vor dem Anhalten der Meldepunkte weitergesendet werden, um die Fehlersituation reproduzieren zu können.
- Der Replay-Funktion soll unendlich oft wiederholt werden können.

9.2 Lösungskonzept 1

Das Lösungskonzept eins ordnet die Replay-Funktion, wie in Abbildung 84 gezeigt wird, der Anwendung zu. Der Datenaustausch mit der Benutzeroberfläche findet dabei über die Datenbank statt. Informationen über die vom Benutzer ausgelöste Replay-Funktion sollen der Anwendung des Emulators über eine eigene Tabelle der Datenbank zugänglich gemacht werden. Die Tabelle speichert den gewählten Zeitpunkt sowie die betroffenen Meldepunkte des Replays.

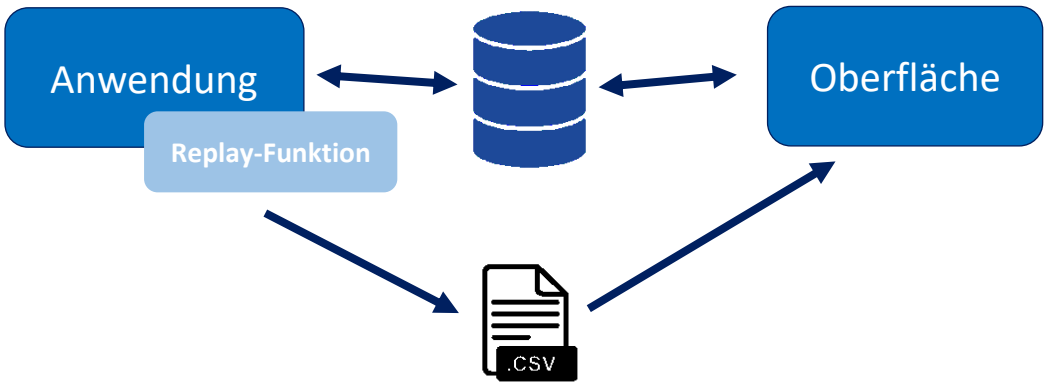


Abbildung 84: Architektureingliederung Replay-Funktion Lösungskonzept 1, Quelle: Eigene Darstellung

Die in Kapitel 9.1 erwähnte „Pause“ Schaltfläche soll dazu führen, dass alle ausgewählten Meldepunkte in den Status „Hold“ wechseln. Zusätzlich sollen alle Bewegungen der Meldepunkte aus den Ereignislogfiles geladen und in einer Tabelle angezeigt werden. Über ein Eingabefeld soll der Benutzer dann den Zeitpunkt wählen können, zu dem zurückgesprungen werden soll. Sobald die Eingabe erfolgt ist, soll für jeden betroffenen Meldepunkt ein Eintrag in der Replay-Tabelle der Datenbank angelegt werden.

Sobald der Bediener alle Ladehilfsmittel manuell hinter die Meldepunkte gebracht hat, kann die Wiederholung der Situation über eine „Wiedergabe“ Schaltfläche gestartet werden. Alle ausgewählten Meldepunkte werden von der Oberfläche damit in den Status „Replay“ gebracht. Die Zuordnung der Ladehilfsmittel zu ihrer aktuellen Route oder Meldepunkt laut Datenbank werden dabei nicht aktualisiert. Die Meldepunkte müssen aus diesem Grund die gleichen Wegentscheidungen treffen, die zuvor im Ereignislogfile protokolliert wurden.

Der Benutzer kann wie bereits erwähnt einen Zeitpunkt wählen, an dem die Replay-Funktion starten soll. Eingehende Transportanfragen werden aber wie im originalen Ablauf so lange zurückgehalten, bis sie laut Zeitstempel des LHM aus dem Ereignislogfile passieren dürfen.

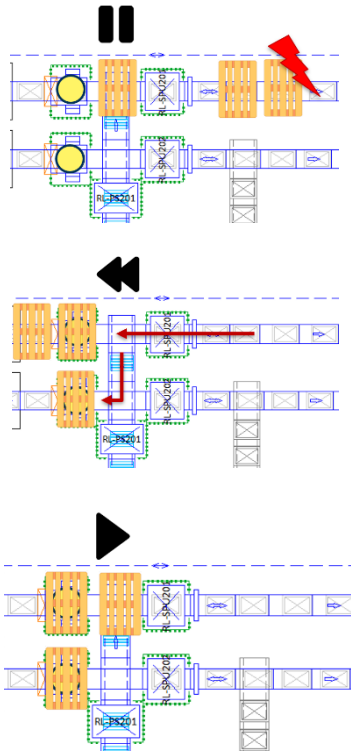


Abbildung 85: Darstellung der LHM Bewegungen während der Replay-Funktion, Quelle: Eigene Darstellung

Sobald alle LHM, die von der Replay-Funktion betroffen sind, über den Meldepunkt gefahren sind, wechselt dieser selbstständig in den Status „Ready“ zurück.

Vorteile:

Die Vorteile dieses Lösungskonzepts sind:

- Der neue „Replay“ Status zeigt weiteren Bedienern, welcher Meldepunkt aktuell die Replay-Funktion ausführt.
- Die LHM fahren immer exakt den gleichen Weg wie in der Störsituation.
- Da die Meldepunkte selbst die Zeitabstände überwachen, können diese genau eingehalten werden.

Nachteile:

Die Nachteile dieses Lösungskonzepts sind:

- Starker Eingriff der Funktion in bestehende Klassen der Meldepunkte
- Aufteilung der Funktionen auf Oberfläche und Anwendung
- Erweiterung der Datenbank um eine weitere Tabelle

9.3 Lösungskonzept 2

Die Herangehensweise des zweiten Lösungskonzepts an die Problemstellung unterscheidet sich grundlegend vom ersten Konzept. Die Gegensätzlichkeit liegt in der Zuordnung der Funktion zur Oberfläche, wodurch die Replay-Funktion vollständig abgekapselt von der Anwendung entwickelt werden kann. Die Abbildung 86 zeigt die Eingliederung in die Architektur des Emulators.

Eine eigene Replay-Klasse, die im ViewModel der Seite instanziiert werden kann, soll alle benötigten Funktionen beinhalten. Wie schon im ersten Lösungskonzept bringt die Klasse alle ausgewählten Meldepunkte nach dem Betätigen des „Pause“ Buttons in den Status „Hold“. Anders als in Lösungskonzept eins werden dabei alle von der Funktion betroffenen Ladehilfsmittel auf den Meldepunkt zurückgebucht. Die in der Datenbank gespeicherte Position der LHM stimmt, nachdem der Bediener diese wieder händisch hinter die Meldepunkte gebracht hat, mit der physikalischen Position überein.

Nach der Wahl des Replay Zeitpunktes und dem Start der Wiedergabe durch den Bediener werden die Meldepunkte von der Replay-Klasse automatisch in den gewünschten Zeitabständen auf „Ready“ bzw. „Hold“ geschalten. Auf „Ready“ sollen die Meldepunkte wechseln, wenn die Zeit zwischen den LHM, die aus dem Ereignisprotokoll des Meldepunktes entnommen werden kann, abgelaufen ist. Sobald ein neuer Eintrag im Logfile hinzukommt, soll der Meldepunkt wieder in den Status „Hold“ schalten. Dieser Vorgang wird so lange wiederholt, bis alle von der Replay-Funktion betroffenen LHM über den Meldepunkt gefahren sind.

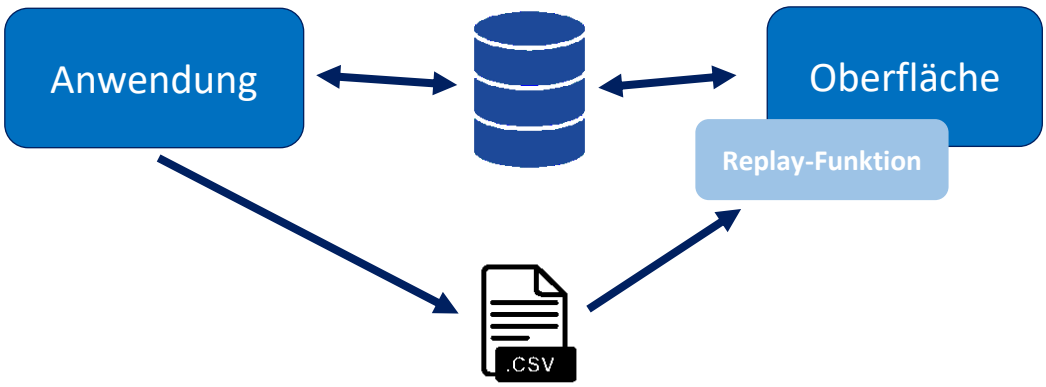


Abbildung 86: Architektureingliederung Replay-Funktion Lösungskonzept 2, Quelle: Eigene Darstellung

Vorteile:

Die Vorteile dieses Lösungskonzepts sind:

- Die Funktion kann abgekapselt von der Anwendung entwickelt werden.
- Es ist kein zusätzlicher Code in der Anwendung und keine zusätzliche Tabelle in der Datenbank notwendig.

Nachteile:

Die Nachteile dieses Lösungskonzepts sind:

- Der aktuelle Status des Meldepunktes gibt keine Auskunft über die Aktivität der Replay-Funktion.

- Da die Ladehilfsmittel auf den vorigen Meldepunkt zurückgebucht werden, wird auch die Route zum nächsten neu berechnet. Es kann daher nicht mit Sicherheit davon ausgegangen werden, dass der gleiche Weg gewählt wird.
- Da von der Statusänderung des Meldepunktes auf „Ready“ bis zum Senden des Transportauftrages eine Sekunde vergehen kann, gilt dieses Konzept als sehr ungenau.

9.4 Auswahl des Lösungskonzeptes

Auch wenn das zweite Lösungskonzept eine vollständig abgekapselte Entwicklung der Replay-Funktion ermöglicht, ist das erste Lösungskonzept geeigneter für diese Anwendung. Die Nachteile des zweiten Konzepts, wie die zeitliche Ungenauigkeit und die Unsicherheit, ob das LHM die gleiche Route wie im Störfall wählt, machen es zu fehleranfällig für den gewünschten Verwendungszweck.

Trotz der nötigen Erweiterung der Datenbank um eine Tabelle überwiegen die Vorteile des ersten Lösungskonzeptes, weshalb es zur Lösung der definierten Problemstellung eingesetzt werden soll.

9.5 Schwierigkeiten des Konzepts in der praktischen Umsetzung

Während der Entwicklung der Lösungskonzepte wurden mehrere Schwierigkeiten in der praktischen Durchführung der Replay-Funktion sichtbar. Ziel dieses Unterkapitels ist es, diese zu dokumentieren und zu entscheiden, ob die Umsetzung im Rahmen der verfügbaren Ressourcen des Projektes möglich und sinnvoll ist.

Bereits während der Definition der Anforderungen an die Replay-Funktion wurde sichtbar, dass manche Vorgänge des Fördersystems nicht rückgängig gemacht werden können und daher von der Funktion ausgenommen werden müssen. In der Konzeptionierung wurde weiters festgestellt, dass sich diese Sonderfälle nicht nur auf Abnahmepunkte beschränken, sondern auch Einlagerungs- und Auslagerungsstationen der Regalbediengeräte einschließen, weil sie maßgeblich von den Aufnahme- und Abgabezeitpunkten des RBG abhängen.

Während der Entwicklung der Konzepte wurde ein zweites Problem erkannt, das sich aus der Anzahl der Meldepunkte in der Anlage ergibt. Für die Nutzung der Replay-Funktion ist eine hohe Dichte an Meldepunkten innerhalb der Anlage von Vorteil. Eine geringe Anzahl an Meldepunkten macht es umständlich und zeitaufwändig, Ladehilfsmittel über weite Strecken im Handbetrieb bis zum vorherigen Meldepunkt zu bringen. Die Abbildung 87 zeigt, dass es bei einer geringen Dichte an Meldepunkten – wie in der Mitte der Anlage – nur schwer möglich ist die Replay-Funktion in der Praxis anzuwenden.

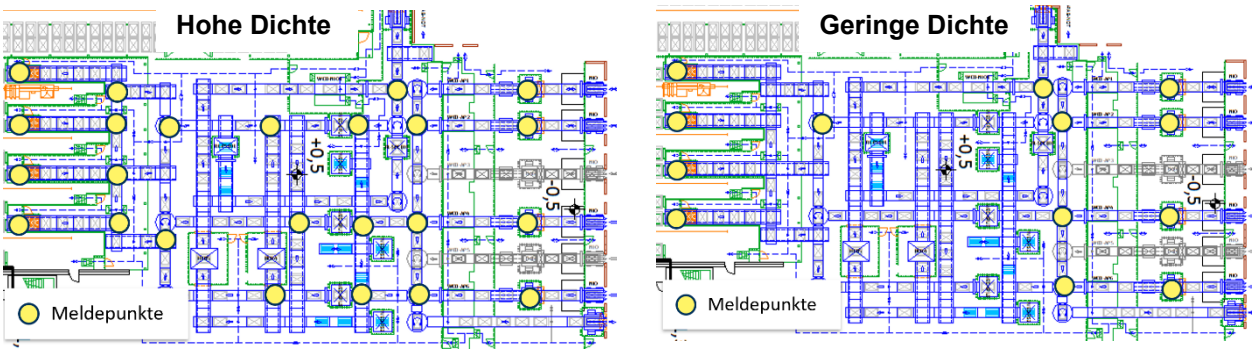


Abbildung 87: Vergleich hohe Dichte zu einer geringen Dichte an Meldepunkten, Quelle: Eigene Darstellung

Eine weitere Problemsituation stellt eine hohe Auslastung der Anlage dar. Die Staubildung hinter einem Störfall macht es, wie in Abbildung 88 ersichtlich, für den Bediener schwer bis nicht möglich, die Ladehilfsmittel im Handbetrieb hinter den nächsten Meldepunkt zu fahren.

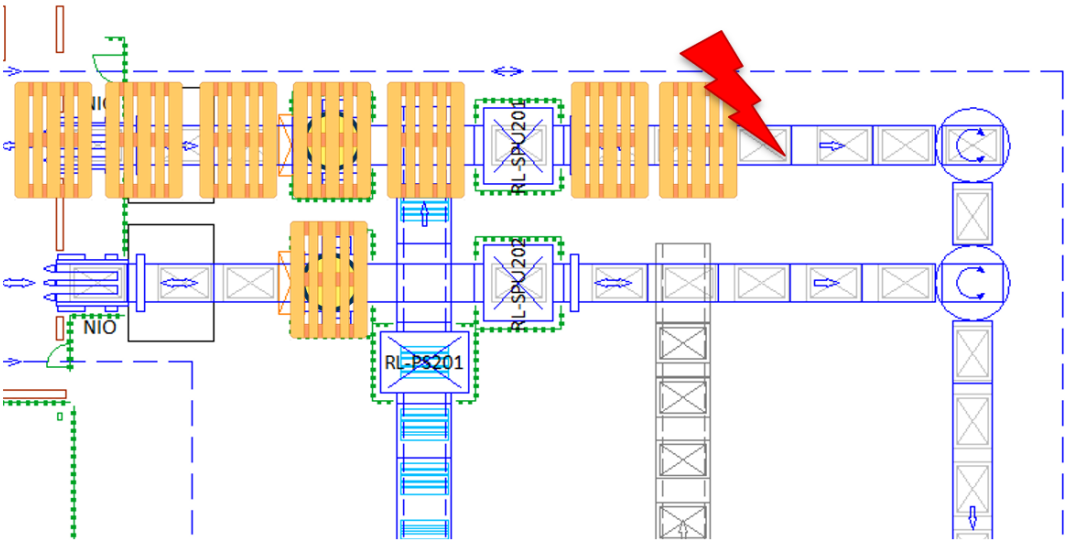


Abbildung 88: Problemsituation bei Staubildung hinter dem Störfall, Quelle: Eigene Darstellung

Alle drei Problemfälle der praktischen Umsetzung wurden mit den Entscheidungsträgern von Jungheinrich Systemlösungen besprochen. Dabei wurde entschieden, dass die Funktion dennoch entwickelt werden soll, da der Aufwand für die technische Umsetzung des ersten Lösungskonzeptes auf weniger als 50 Stunden geschätzt wird und die Funktion im Einzelfall doch einen erheblichen Mehrwert für die Inbetriebnahme schaffen kann.

10 ERGEBNISSE UND AUSBLICK

Ziel der Arbeit ist es, ein Konzept zu erstellen, das die Entwicklung eines Warehouse-Control-System Emulators für die Inbetriebnahme von Logistikanlagen ermöglicht. Dieser soll bei der Inbetriebnahme der Anlage unterstützen und ihre Funktion nachweisen können. Zusätzlich wird eine Testfunktion programmiert, mit deren Hilfe die Leistung von Regalbediengeräten und Fördersystemen überprüft werden kann.

Zu Beginn der Arbeit werden die Anforderungen, die Grenzen und die zu behandelnden Funktionen des Emulators und der zu emulierenden Warehouse-Management-Systeme definiert. Diese sind vorrangig das SAP-EWM-System von LIDL und das Jungheinrich-WMS. Weitere benötigte WMS werden in den nächsten Jahren hinzugefügt.

Für die automatische Erstellung der in der Datenbank abgelegten Konfigurationsdaten soll eine Parameterdatei angelegt werden, deren Auszeichnungssprache innerhalb dieser Arbeit evaluiert wird. Zur Auswahl stehen XML, JSON und YAML, wobei JSON in den Kriterien Lesbarkeit, Zusammenspiel mit C# und Overhead bei der Nutzwertanalyse am besten abschneidet. Die Parameterdatei soll gemeinsam mit dem SPS-Programm der Steuerung automatisch von einem bestehenden Tool der Steuerungstechnikabteilung bei Jungheinrich Systemlösungen generiert werden.

Während der Planung der Softwarearchitektur wird auch das Vorgehensmodell zur Softwareentwicklung definiert. Dieses soll die Vorteile des V-Modells mit denen der agilen Modelle vereinen und so ein hybrides Vorgehensmodell bilden. Der Ablauf der Entwicklungsphase wird durch Einsatz dieses Modells in mehrere Sprints geteilt, wobei der erste Sprint eine verlängerte Analyse- und Designphase vorsieht, in der die allgemeinen Anforderungen und das Grunddesign definiert werden. Teil der Softwarearchitektur-Planung ist es, zusätzlich das System in einzelne Pakete zu teilen, um diese getrennt voneinander entwickeln zu können. Zu diesen Paketen zählen die Datenbank, die Anwendung und die Bedienoberfläche. Für jeden dieser drei Teile werden getrennt Anforderungen definiert und davon abgeleitete Lösungskonzepte entwickelt. Im Rahmen dieser Lösungskonzepte sollen möglichst früh in der Konzeptionierung des Systems Prototypen erstellen werden, um auftretende Schwierigkeiten zu erkennen und zu dokumentieren.

Neben der Konzepterstellung des Emulators ist auch die Entwicklung einer automatischen Leistungstestfunktion Bestandteil der vorliegenden Arbeit. Die Funktion ist für Fördersysteme und Regalbediengeräte getrennt zu behandeln. Das Konzept für ihre Programmierung wird von den definierten Anforderungen abgeleitet und umgesetzt. Nach dem Erstellen beider Teilfunktionen werden diese durch einen Funktionstest überprüft. Für beide Tests liegt ein positives Resultat vor.

Der letzte Teil dieser Masterarbeit beschäftigt sich mit der Replay-Funktion des Emulators. Diese soll den Inbetriebnehmern helfen, Fehlerfälle nachzustellen, um dadurch Probleme einfacher beheben zu können. Für die Umsetzung gibt es zwei von den Anforderungen abgeleitete Lösungskonzepte. Das erste Konzept ordnet die Funktion der Anwendung zu, wobei sie direkt in den Klassen der Applikation enthalten ist und somit besonders genau die Fehlersituation nachbilden kann. Es wird dem zweiten Konzept, das von der Anwendung abgekapselt ist, vorgezogen, da bei diesem die Nachteile wie die zeitliche Ungenauigkeit und hohe Störanfälligkeit überwiegen.

Das entwickelte Softwarekonzept, die Datenbank, das Framework der Benutzeroberfläche, das bereits getestete Leistungstest-Modul sowie das Konzept für die Umsetzung der Replay-Funktion dienen als Grundlage für das weitere Vorgehen. Diese Umsetzung soll von Jänner 2019 bis April 2019 erfolgen, wobei aktuell überlegt wird, die Entwicklung von einem externen Unternehmen durchführen zu lassen. Grund dafür ist, dass Jungheinrich Systemlösungen im Jahr 2019 nicht die nötigen Ressourcen bereitstellen kann, um das Projekt umzusetzen. Das hybride Vorgehensmodell soll auch bei Fremdvergabe des Entwicklungsprojekts zum Einsatz kommen. Ein besonderer Wert wird dabei auf die zweiwöchentlichen Sprintmeetings und auf den einwöchigen Workshop am Beginn des Projekts gelegt. Teil der Sprintmeetings soll das Besprechen der weiteren Vorgehensweise und weiterer Arbeitspakete sein sowie ein Review des letzten Sprints. Im einwöchigen Workshop soll es zum Wissenstransfer zwischen Jungheinrich Systemlösungen und dem beauftragten Unternehmen kommen, wobei die vorliegende Arbeit einen entscheidenden Beitrag leisten soll.

Die Testphase des WMS-Emulators startet Ende April 2019 und soll in einer virtuellen Umgebung erfolgen. Die Hardware wird während der Tests von der Software Emulate3D emuliert und die speicherprogrammierbare Steuerung wird von der Software PLCSim Advanced simuliert. Ende August 2019 soll der entwickelte Emulator dann während der ersten Inbetriebnahmephase eines Großprojektes eingesetzt werden.

LITERATURVERZEICHNIS

Gedruckte Werke (12)

Federation Europeenne de la Manutention (Hrsg.) (2003): *Leistungsnachweis für Regalbediengeräte Spielzeiten*, Deutschland

Beck, Kent; Fowler, Martin (2001): *Extreme Programming planen*, 1. Auflage, Addison-Wesley Verlag, München

Dieter, Arnold; Furmanns, Kay (2009): *Materialfluss in Logistiksystemen*, 6. Auflage, Springer-Verlag Berlin Heidelberg, London; New York

Geißen, Tim ; Bodden-Streubühr, Michael; Geldmacher, Fin; Ludwigs, Helmut; Pfisterer, Günter; Pott, Christoph; Spee, Detlef (2014): *VDI 3601*, Beuth Verlag GmbH, Düsseldorf

Gloger, Boris (2008): *Scrum - Produkte zuverlässig und schnell entwickeln*, Carl Hanser Verlag, München

Lerman, Julia (2010): *Programming Entity Framework*, 2. Auflage, Sebastopol

McLean Hall, Gary (2010): *Pro WPF and Silverlight MVVM* , 1. Auflage, Apress

Noorul, Hassan; Ahmed, Kamal; Shirazi, Sharif; Yusop, Zulkifli (2015): *Weighting Methods and their Effects on Multi-Criteria Decision Making Model Outcomes in Water Resources Management*, Springer

Sandhaus, Gregor; Berg, Björn; Knott, Philip (2014): *Hybride Softwareentwicklung*, 1. Auflage, Springer, Berlin

Skulschus, Marco; Wiederstein, Marcus; Winterstone, Sarah (2011): *XML Schema*, Cornelio, Berlin

Smith, Josh (2010): *Advanced MVVM*

Wieczorrek, H.W.; Mertens, P. (2008): *Management von IT-Projekten*, 3. Auflage, Springer, Berlin

Online-Quellen (8)

json (2018): *json*
www.json.org [Stand: 16.07.2018]

w3school (2018): *w3school*
https://www.w3schools.com/js/js_json_xml.asp [Stand: 16.07.2018]

YAML (2018): *www.yaml.org*
http://yaml.org/spec/1.2/spec.html [Stand: 16.07.2018]

Microsoft (2016): *Docs*
https://docs.microsoft.com/de-de/ef/core/ [Stand: 02.09.2018]

Beck, Kent; Beedle, Mike; van Bennekum, Arie; Cockburn, Alistair; Cunningham, Ward; Fowler, Martin; Grenning, James; Highsmith, Jim; Hunt, Andrew; Jeffries, Ron; Kern, Jon; Marick, Brian; Martin, Robert; Mellor , Steve; Schwaber, Ken; Sutherland, Jeff; Thomas, Dave; Schwaber, Ken (2001): *agilemanifesto*
http://agilemanifesto.org/iso/de/manifesto.html [Stand: 01.08.2018]

Kersken, Sascha (2004): *openbook.rheinwerk-verlag*
http://openbook.rheinwerk-verlag.de/kit/itkomp15000.htm#Xxx999149 [Stand: 16.07.2019]

Klaus, Vitt (2018): *www.cio.bund.de*
https://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT/vmodell_xt_node.html
[Stand: 26.07.2018]

Quin, Liam (2018): *W3C*
https://www.w3.org/standards/xml/core [Stand: 16.07.2018]

ABBILDUNGSVERZEICHNIS

Abbildung 1: Zeitlicher Vergleich der Inbetriebnahmephase mit und ohne WMS Emulator, Quelle: Eigene Darstellung 3

Abbildung 2: Kern- und Zusatzfunktionen eines WMS, Quelle: VDI 3601, S. 6. 4

Abbildung 3: Eigliederung des SAP-EWM in die SAP Systemlandschaft, Quelle: Eigene Darstellung 5

Abbildung 4: Basismodule Jungheinrich-WMS, Quelle: Eigene Darstellung 7

Abbildung 5: Materialflussschema, Quelle: Eigene Darstellung. 11

Abbildung 6: Ladeeinheiten auf einer Förderstrecke der Länge l, Quelle: Eigene Darstellung 11

Abbildung 7: Teilstetig Verzweigung für 2 Richtungen, Quelle: Eigene Darstellung 12

Abbildung 8: Aufbau des Arbeitsspieles eines Regalbediengerätes, Quelle: Eigene Darstellung 13

Abbildung 9: Mittleres Einzelspiel Fall 1, Quelle: Eigene Darstellung 14

Abbildung 10: Mittleres Doppelspiel Fall 1, Quelle: Eigene Darstellung 14

Abbildung 11: Mittleres Einzelspiel Einlagerung Fall 5, Quelle: Eigene Darstellung 15

Abbildung 12: Mittleres Einzelspiel Auslagerung Fall 5, Quelle: Eigene Darstellung 15

Abbildung 13: Mittleres Doppelspiel Fall 5, Quelle: Eigene Darstellung 15

Abbildung 14: Codebeispiel XML, Quelle: Eigene Darstellung 17

Abbildung 15: Codebeispiel JavaScript Object Notation, Quelle: Eigene Darstellung 18

Abbildung 16: Codebeispiel YAML Ain‘t no Markup Language, Quelle: Eigene Darstellung 18

Abbildung 17: Entwicklertool der Steuerungstechnik bei Jungheinrich, Quelle: Eigene Darstellung 19

Abbildung 18: V-Modell, Quelle: Eigene Darstellung 21

Abbildung 19: Vergleich phasenorientiertes Vorgehensmodell und hybrides Vorgehensmodell, Quelle: Eigene Darstellung 24

Abbildung 20: MVVM Konzept, Quelle: Eigene Darstellung 25

Abbildung 21: Abfragen von Daten aus einer Datenbank mit dem Entity Framework, Quelle: Eigene Darstellung 26

Abbildung 22: Speichern von Daten in einer Datenbank mit dem Entity Framework, Quelle: Eigene Darstellung 26

Abbildung 23: Systemteile des Emulators, Quelle: Eigene Dartstellung 28

Abbildung 24: LHM Tabelle, Quelle: Eigene Dartstellung 30

Abbildung 25: Tabelle Plc, Quelle Eigene Darstellung 30

Abbildung 26: Tabelle ReportingPoint, Quelle: Eigene Darstellung 31

Abbildung 27: Tabelle ReportingPointLoc, Quelle: Eigene Darstellung 31

Abbildung 28: Tabelle WhPos, Quelle: Eigene Darstellung 31

Abbildung 29: Tabelle Connection: Quelle: Eigene Darstellung 31

Abbildung 30: Tabelle Route, Quelle: Eigene Darstellung 32

Abbildung 31: Tabelle Order, Quelle: Eigene Darstellung 32

Abbildung 32: Tabelle VehicleOrders, Quelle: Eigene Darstellung 32

Abbildung 33: Tabelle ErrorMsg, Quelle: Eigene Darstellung 32

Abbildung 34: Tabelle Error, Quelle: Eigene Darstellung 33

Abbildung 35: Aufbau und Beziehungen aller Tabellen der Datenbank, Quelle: Eigene Darstellung 33

Abbildung 36: Entitätsmodell Visual Studio, Quelle: Eigene Darstellung 36

Abbildung 37: Prototyp Entity Framework Versuch 1, Quelle: Eigene Darstellung 37

Abbildung 38: Prototyp Entity Framework Versuch 2, Quelle: Eigene Darstellung 37

Abbildung 39: Architekturübersicht Anwendungssoftware, Quelle: Eigene Darstellung 39

Abbildung 40: Kommunikations-Klassen, Quelle: Eigene Darstellung 40

Abbildung 41: Klasse RepPoint, Quelle: Eigene Darstellung 42

Abbildung 42: Meldepunkt - Klassen, Quelle: Eigene Darstellung 43

Abbildung 43: Klasse IPoint, Quelle: Eigene Darstellung 43

Abbildung 44: Klasse OPoint, Quelle: Eigene Darstellung 43

Abbildung 45: Klasse PsPoint, Quelle: Eigene Darstellung 44

Abbildung 46: Klasse RsPoint, Quelle: Eigene Darstellung 44

Abbildung 47: Klasse StcLam, Quelle: Eigene Darstellung 44

Abbildung 48: Anordnung der Lastaufnahmemittel auf einem Regalbediengerät, Quelle: Eigene Darstellung 45

Abbildung 49: Zielsuche Klasse Route, Quelle: Eigene Darstellung 46

Abbildung 50: Lagerplatzsuche, Quelle: Eigene Darstellung 47

Abbildung 51: Wegstrecke vom Lagerplatz zur Versandbahn, Quelle: Eigene Darstellung 48

Abbildung 52: Aufteilung des Hauptfensters, Quelle: Eigene Darstellung 49

Abbildung 53: Begrüßungsbildschirm WMS Emulator, Quelle: Eigene Darstellung 50

Abbildung 54: Beispiel für das Design eines Eingabedialoges, Quelle: Eigene Darstellung 50

Abbildung 55: Einsatz von Tabellen und Dropdown Feldern in der Benutzeroberfläche, Quelle: Eigene Darstellung 52

Abbildung 56: Teile des MVVM, Quelle: Eigene Darstellung 53

Abbildung 57: Datagrid in WPF, Quelle: Eigene Darstellung 53

Abbildung 58: Combobox in WPF, Quelle: Eigene Darstellung 54

Abbildung 59: Klasse RelayCommand, Quelle: Eigene Darstellung 54

Abbildung 60: Click Command WPF, Quelle: Eigene Darstellung 55

Abbildung 61: Wechesln des Inhaltsbereiches des Hauptfensters, Quelle; Eigene Darstellung 55

Abbildung 62: Versuch 1 Early Prototyping Benutzeroberfläche, Quelle: Eigene Darstellung 56

Abbildung 63: Versuch 2 Early Prototyping Benutzeroberfläche, Quelle: Eigene Darstellung 56

Abbildung 64: Einordnung des Leistungstests in die Architektur des Emulators, Quelle: Eigene Darstellung 58

Abbildung 65: Datenformat Ereignislogfile Fördertechnik, Quelle: Eigene Darstellung 59

Abbildung 66: Datenformat Ereignislogfile Lastaufnahmemittel, Quelle: Eigene Darstellung 59

Abbildung 67: Telegrammquittierung eines vom Emulator gesendeten Telegrammes, Quelle: Eigene Darstellung 60

Abbildung 68: Dialog zum Erstellen der Fahraufträge des Leistungstests, Quelle: Eigene Darstellung ... 61

Abbildung 69: Prüfen der Aufnahmeposition bei der Leistungstestvorbereitung, Quelle: Eigene Darstellung 61

Abbildung 70: Visualisierung der Vorbereitungsphase anhand des Start Button, Quelle: Eigene Darstelung 62

Abbildung 71: Auszug Ereignislogfile RBG Jungheinrich-WMS, Quelle: Eigene Darstellung 62

Abbildung 72: Auszug Ereignislogfile RBG SAP-EWM, Quelle: Eigene Darstellung 62

Abbildung 73: Liniendiagram Leistungstest RBG, Quelle: Eigene Darstellung 63

Abbildung 74: Gauge-Diagramm Leistungstest RBG, Quelle: Eigene Darstellung 64

Abbildung 75: Leistungstest RBG Bedienoberfläche, Quelle: Eigene Darstellung 64

Abbildung 76: Bedienoberfläche Leistungstestvergleich RBG, Quelle: Eigene Darstellung 65

Abbildung 77: Dialog zum Erstellen des Fördertechnik Leistungstests, Quelle: Eigene Darstellung 65

Abbildung 78: Vorbereitung Leistungstest Fördertechnik, Quelle: Eigene Darstellung 66

Abbildung 79: Anzeigen der Daten des Logfiles in der Benutzeroberfläche, Quelle: Eigene Darstellung . 67

Abbildung 80: Dynamische Anzeige der Leistung des Meldepunktes, Quelle: Eigene Darstellung 67

Abbildung 81: Leistungstest Fördertechnik Bedienoberfläche, Quelle: Eigene Darstellung 68

Abbildung 82: Ablauf der Überprüfung der RBG Leistungstest Anwendung, Quelle: Eigene Darstellung 69

Abbildung 83: Ablauf der Überprüfung der Fördertechnik Leistungstest Anwendung, Quelle: Eigene Darstellung 70

Abbildung 84: Architektureingliederung Replay-Funktion Lösungskonzept 1, Quelle: Eigene Darstellung71

Abbildung 85: Darstellung der LHM Bewegungen während der Replay-Funktion, Quelle: Eigene Darstellung 72

Abbildung 86: Architektureingliederung Replay-Funktion Lösungskonzept 2, Quelle: Eigene Darstellung73

Abbildung 87: Vergleich hohe Dichte zu einer geringen Dichte an Meldepunkten, Quelle: Eigene Darstellung 74

Abbildung 88: Problemsituation bei Staubildung hinter dem Störfall, Quelle: Eigene Darstellung 75

TABELLENVERZEICHNIS

Tabelle 1: Vergleich Jungheinrich-WMS und SAP-EWM, Quelle: Eigene Darstellung 8

Tabelle 2: Evaluierung Extensible Markup Language, Quelle: Eigene Darstellung 17

Tabelle 3: Evaluierung JavaScript Object Notation, Quelle: Eigene Darstellung 18

Tabelle 4: Evaluierung YAML Ain‘t no Markup Language, Quelle: Eigene Darstellung 19

Tabelle 5: Vorgehensmodelle zur Softwareentwicklung, Quelle: Eigene Darstellung 20

Tabelle 6: Beziehungen Tabelle ReportingPoint, Quelle: Eigene Darstellung 34

Tabelle 7: Beziehungen Tabelle Plc: Quelle: Eigene Darstellung 34

Tabelle 8: Beziehungen Tabelle ErrorMsg, Quelle: Eigene Darstellung 35

Tabelle 9: Beziehungen Tabelle ReportingPoint, Quelle: Eigene Darstellung 35

ABKÜRZUNGSVERZEICHNIS

CSV	Comma-Separated-Value
ERP	Enterprise-Resource-Planning
EWM	Extended-Warehouse-Management
XML	Extensible Markup Language
XP	Extreme Programming
XT	Extreme Tailoring
FEM	Fédération Européenne de la Manutention
HU	Handling Unit
ID	Identifikationsnummer
JSON	JavaScript Object Notation
JSL	Jungheinrich Systemlösungen GmbH
LE	Ladeeinheit
LINQ	Language Integrated Query
LAM	Lastaufnahmemittel
MFS	Materialflusssteuerungssystemen
MVVM	Model View ViewModel
NiO	Nicht in Ordnung
ORM	Object Relational Mapping
PLC	Programmable Logic Controller
QM	Qualitätsmanagement
QIE	Quality Inspection Engine
RBG	Regalbediengerät
SPS	Speicherprogrammierbare Steuerung
SGML	Standard Generalized Markup Language
ST	Steuerungstechnik
SCM	Supply-Chain-Management
UI	User Interface
WCS	Warehouse-Control-System
WMS	Warehouse-Management-System
WPF	Windows Presentation Foundation

W3C	World-Wide-Web-Konsortium
YAML	YAML Ain’t Markup Language