Bachelor-Thesis

Analyse und Konzeption einer benutzerfreundlichen GUI zur Vereinfachung der Administration einer Schnittstelle für die Integration von Unternehmenssystemen

Vorgelegt am: 09.08.2024

Von: **Mohamad Zakaria Enabi**
Reitzenhainer Str. 7
09120 Chemnitz

Studiengang: Wirtschaftsinformatik

Studieneinrichtung: Wirtschaftsinformatik

Seminargruppe: 4WI21-1

Matrikelnummer: 4004601

Praxispartner:

iFD GmbH
Schulstraße 38
09125 Chemnitz

Gutachter: Ansgar Fischer (iFD GmbH)

Prof. Dr. Mathias Sporer (Staatliche Studienakademie Glauchau)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
DTD	Document Type Definition
ERP	Enterprise Resource Planning
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IoC	Inversion of Control
JSON	Java Script Object Notation
JWT	JSON Web Token
OAuth2	Open Authorization 2.0
ORM	Object-Relational-Mapping
REST	Representational State Transfer
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSMS	SQL Server Management Studio
TCP	Transmission Control Protocol
WMS	Warehouse Management System
XML	Extensible Markup Language

1 Einleitung

1.1 Vorstellung

Die iFD GmbH ist global tätig und spezialisiert sich auf die Konzeption, Entwicklung und Umsetzung hochmoderner Softwaresysteme für automatisierte und manuelle Anwendungen in der Intralogistik, die in Industrie- und Handelsunternehmen zum Einsatz kommen.

Das Leistungsspektrum umfasst eine Vielzahl von Anwendungen, darunter Lagerverwaltungssysteme, präzise Materialflusssteuerungen, effiziente Staplerleitsysteme sowie fortschrittliche Simulationen. Diese Systeme werden anschließend durch kundenspezifische Module und Prozesse maßgeschneidert erweitert, um jedem Kunden eine optimale und bedarfsgerechte Lösung bieten zu können.

Im heutigen globalisierten Geschäftsumfeld bilden gut integrierte Softwaresysteme das Rückgrat für reibungslose betriebliche Abläufe. Als eine IT-Firma, die sich auf Softwaresysteme für die Intralogistik spezialisiert hat, trägt die iFD GmbH maßgeblich dazu bei, Unternehmen wettbewerbsfähiger zu machen sowie deren betriebliche Abläufe effizienter und zukunftsfähiger zu gestalten. Dazu gehört auch die Berücksichtigung des Prinzips der ökonomischen Minimierung, bei dem die für die Fertigstellung eines Kundenprojekts benötigten Entwicklungsstunden so weit wie möglich reduziert werden, um das Kundenprojekt schnellstmöglich und fehlerfrei in Betrieb zu nehmen und somit einen höheren Ertrag zu erzielen.

Diese Arbeit konzentriert sich auf die Konzeption und Entwicklung einer benutzerfreundlichen GUI zur Vereinfachung der Administration einer Schnittstelle sowie die Generalisierung einer Datenbank-Schnittstelle, um ein einheitliches Modul für die Kommunikation zwischen dem WMS und externen Unternehmenssystemen wie ERP unabhängig von der Art der Kommunikation zu gewährleisten. Diese Generalisierung bildet die notwendige Voraussetzung für die anschließende Konzeption und Entwicklung dieser benutzerfreundlichen GUI zwischen einem WMS und externen Host-Systemen.

Der Kern dieser Arbeit liegt in der Entwicklung eines innovativen Konzepts, das darauf abzielt, eine benutzerfreundliche Oberfläche für die Konfiguration von Schnittstellen-Informationen zu entwickeln. Dabei soll eine generische Schnittstelle entworfen werden, die den Datenaustausch über verschiedene Wege ermöglicht. Die Nutzung einer solchen Plattform bietet die Möglichkeit, Daten in einem einheitlichen Format zu übertragen, was nicht nur die Konsistenz der Daten sicherstellt, sondern auch zukünftige Schnittstellenprojekte beschleunigt.

Zentrale Aufgabe dieser Arbeit ist die Beantwortung zweier prinzipieller Fragen: Erstens, wie lässt sich eine solche Schnittstelle modular generalisieren, um sie für jede Schnittstellentechnologie einzusetzen und einfach anpassbar zu gestalten?

Zweitens, wie kann eine benutzerfreundliche grafische Benutzeroberfläche (GUI) konzeptioniert und entwickelt werden, die die Verwaltung einer Schnittstelle zur Integration von Unternehmenssystemen vereinfacht?

Die praxisorientierte Ausrichtung dieser Arbeit erlaubt es, reale Probleme im Bereich der Systemintegration anzugehen und kreative Lösungsansätze zu entwickeln. Die Interaktion zwischen WMS und ERP-System ist von zentraler Bedeutung, da sie die Grundlage für einen reibungslosen Datenaustausch schafft.

Die Erkenntnisse und Ergebnisse dieser Bachelor-Thesis werden nicht nur unsere Fähigkeiten in der Systemintegration stärken, sondern auch dazu beitragen, die Effizienz der Geschäftsprozesse im Bereich der Intralogistik zu steigern und den Aufwand für eine neue Schnittstellenentwicklung anderer Projekte zu mindern.

1.2 Problemstellung und Motivation

Aktuell wird in jedem Kundenprojekt die Schnittstellenentwicklung entweder komplett neu durchgeführt oder es werden Schnittstellen von älteren Kundenprojekten übernommen. Außerdem erfolgt die Konfiguration von Schnittstellen in der iFD manuell, was nicht nur einen hohen Zeitaufwand bedeutet, sondern auch eine tiefe technische Expertise voraussetzt. Dies führt zu einer erhöhten Fehleranfälligkeit und verlangsamt die Implementierung der zum Start des Projekts benötigten Schnittstelle.

Durch die mögliche Verlangsamung der gesamten Projektabwicklung könnte die Produktivität, Kundenzufriedenheit und Wettbewerbsfähigkeit des Unternehmens beeinträchtigt werden.

Im Hinblick auf die zunehmende Digitalisierung und der Notwendigkeit für eine rasche und flexible Systemintegration ist es unverzichtbar, effiziente und zuverlässige Lösungen für eine schnelle Inbetriebnahme und übersichtliche Verwaltung von Schnittstellen anzustreben.

Durch die Entwicklung einer generischen und abstrakten Schnittstelle und einer benutzerfreundlichen GUI könnte die iFD erhebliche Zeit- und Kosteneinsparungen realisieren, insbesondere weil die Administration einer Schnittstelle über eine solche GUI keine Entwicklungskenntnisse verlangt. Dadurch könnten beispielsweise auch Projektleiter und Tester die Schnittstelle selbst konfigurieren.

1.3 Zielsetzung der Arbeit

Das Ziel dieser Arbeit besteht in der Implementierung einer abstrakten Schnittstelle sowie in der Konzeption und Entwicklung einer benutzerfreundlichen GUI zur Zuordnung von zu übertragenden Informationen zwischen dem WMS und externen Kundensystemen. Die Entwicklung der GUI setzt voraus, dass die abstrakte Schnittstelle erfolgreich implementiert wird, da sie die Grundlage für das Mapping zwischen den Feldern im System bildet, das über die GUI konfiguriert wird.

Durch die Generalisierung bzw. die Entwicklung einer generischen Schnittstelle sollen Entwicklungsstunden bei der Abwicklung jedes neuen Kundenprojektes verkürzt und eine einheitliche Funktionalität und Struktur der Schnittstellen im Bereich WMS gewährleistet werden. In Kundenprojekten, bei denen eine Kommunikation mit externen Kundensystemen parallel erforderlich ist, sollen lediglich einige Klassen dieser abstrakten Schnittstelle ausimplementiert werden, um kundenindividuelle Informationen zuzuordnen und diese beim Eingang zu erfassen oder beim Ausgang an das Host-System zu senden. Der Kern dieser Schnittstelle sollte dabei unberührt bleiben.

Für die Konzeption dieser Schnittstelle sollen die bisherigen Schnittstellen in den wichtigsten Kundenprojekten auf wiederkehrende Muster geprüft werden, um daraus eine abstrakte und generische Schnittstelle zu entwickeln und in die Basis des WMS zu übernehmen. Diese Schnittstelleentwicklung soll mit jeder Kommunikationsart kompatibel sein und nur mit minimalen unkomplizierten Anpassungen im Kundenprojekt zum Laufen gebracht werden.

Die benutzerfreundliche GUI zur Administration von Schnittstellen soll es den Anwendern ermöglichen, Felder eines bestimmten Objekts, das zwischen dem WMS und dem Host-System übertragen werden soll, zuzuordnen und zu konfigurieren. Dabei sollen Anwender ebenfalls die Möglichkeit haben, die Konvertierung zwischen verschiedenen Datentypen anhand vorgefertigter Standardkonverter vorzunehmen und mehrere Felder von einer Seite auszulesen und in ein bestimmtes Feld auf der Gegenseite zu schreiben. Außerdem soll es auch möglich sein, projektspezifische Konverter zu erstellen und zur Auswahl hinzuzufügen, um alle neuen Anforderungen der Kunden abzudecken. Mithilfe dieser GUI sollen Benutzer die Mapping- und Konvertierungseinstellungen einfach und ohne tiefgehende technische Kenntnisse anpassen können.

2 Grundlagen

2.1 ERP

Ein ERP-System (Enterprise Resource Planning System) ist eine umfassende Software, die die Integration und Automatisierung verschiedener Geschäftsprozesse in Unternehmen unterstützt. Dazu gehören Bereiche wie Finanzen, Produktion, Beschaffung und Personalplanung. Die Bereiche eines ERP-Systems fördern die Datenkonsistenz, Automatisierung und erleichtern die Entscheidungsfindung.

In der heutigen digitalen Welt verfügen Unternehmen über zahlreiche technologische Optionen, um internen Schnittstellen zu definieren und darzustellen. Angesichts des steigenden Wettbewerbsdrucks müssen Firmen rasch auf Informationen zugreifen und sie effizient verarbeiten. Eine Lösung hierfür bietet die Implementierung eines ERP-Systems.[1]

Abbildung 1 Die Integration eines ERP-Systems in Unternehmensbereiche[2]
(sevdesk.de, 27.06.2024)

[1] vgl. online: STEUERWALD, 2017 (27.06.2024)
[2] online: sevdesk, 2024 (27.06.2024)

2.2 WMS

Ein WMS (Warehouse Management System) ist eine Software, die die effiziente Verwaltung von Lager- und Distributionszentren ermöglicht. Das WMS unterstützt Unternehmen bei ihren täglichen Lagerprozessen vom Eingang der Waren und Materialen in einem Lager bis zu deren Versand. Neben der Bestandsführung bietet ein WMS zahlreiche Funktionen wie Wareneingang und -ausgang, Bestandsverfolgung, Auftragsabwicklung, Transportorganisierung, Lagerplatzoptimierung und Versandplanung an.[3]

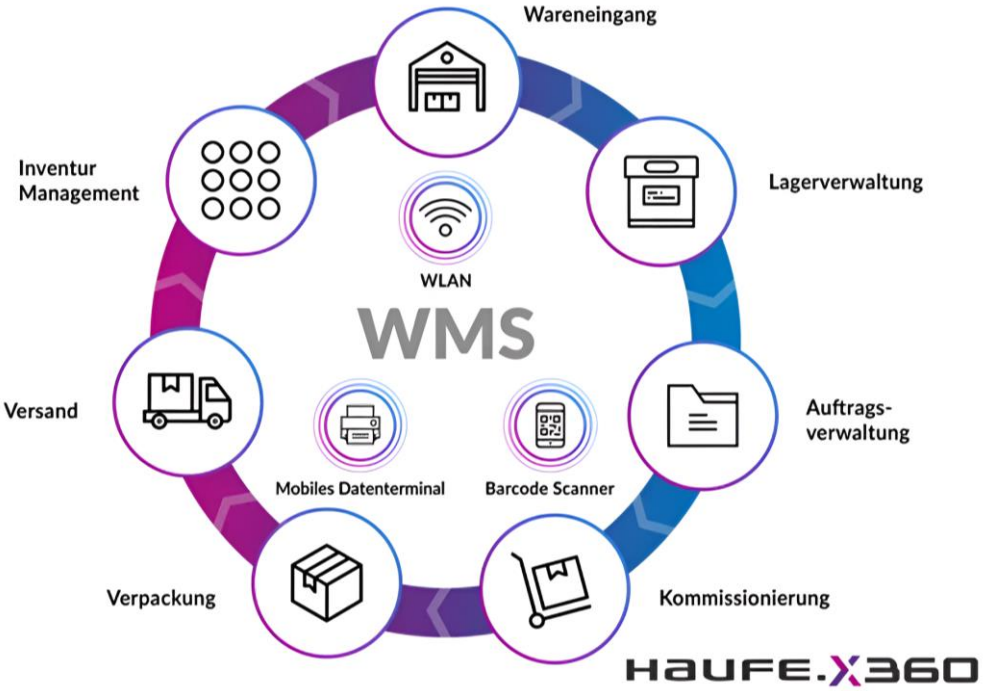


Abbildung 2 Die Aufgaben eines Warehouse Management Systems[4]
(haufe-x360.de, 27.06.2024)

2.3 Unterschied zwischen ERP und WMS

Ein ERP-System zielt darauf ab, alle wesentlichen Geschäftsprozesse wie Finanzen, Personalwesen und Produktion zu automatisieren und bietet eine umfassende Sicht auf das gesamte Unternehmen. Ein WMS hingegen fokussiert sich auf die Echtzeit-Verwaltung von Abläufen im Lager und ist speziell darauf ausgelegt, Lagerprozesse, wie Bestandsverwaltung, Kommissionierung und Versand zu optimieren.[5]

Aufgrund der genannten Unterschiede besteht bei nahezu jedem Kunden die Notwendigkeit, eine reibungslose Integration zwischen ERP-System und WMS zu gewährleisten, was wiederum eine schnelle und effiziente Abwicklung des Kundenpro-

[3] vgl. online: SAP, 2024 (27.06.2024)
[4] online: WMS und ERP: Das sind die Unterschiede, 2024 (27.06.2024)
[5] vgl. online: RÖHRL, 2024 (27.06.2024)

jekts unterstützt. Infolgedessen ist das Thema dieser Arbeit von entscheidender Bedeutung für eine erfolgreiche Umsetzung und Integration der Systeme im Lagermanagement.

2.4 Grafische Benutzeroberfläche (GUI)

Eine GUI (Graphical User Interface, auf Deutsch grafische Benutzeroberfläche) ist eine durch Programme erzeugte visuelle Darstellung von Daten und Zuständen, die es Nutzern ermöglicht, über grafische Elemente wie Symbole, Menüs und Fenster mit einem System zu interagieren. Sie macht die Nutzung von Computern einfacher und benutzerfreundlicher, indem sie Textbefehle durch visuelle Elemente ersetzt. Durch die GUI können Anwender Programme starten, Daten bearbeiten, Einstellungen ändern und viele weitere Aufgaben erledigen.[6]

GUIs bieten viele Vorteile: Sie sind meist benutzerfreundlicher als textbasierte Oberflächen, da sie verständliche Symbole und visuelle Elemente nutzen. Dadurch können auch weniger technisch versierte Nutzer Computer und Software effektiv bedienen.[7]

Einige Nachteile sind der höhere Ressourcenverbrauch, der die Leistung beeinträchtigen kann, und die aufwendige Entwicklung, da sowohl Design als auch Funktionalität berücksichtigt werden müssen.[8]

Trotz dieser Nachteile wurde eine grafische Benutzeroberfläche (GUI) gewählt, da sie die Verwaltung der Schnittstelle auch für Benutzer ohne technische Kenntnisse bedienbar macht.

2.5 Moderne Kommunikationsmethoden

2.5.1 REST-API

2.5.1.1 Definition

Um die Interaktion zwischen unterschiedlicher Software zu erleichtern, greift man häufig auf eine REST-API zurück. REST-API steht für "Representational State Transfer Application Programming Interface" und nutzt Hypertext Transfer Protocol (HTTP) sowie (Hypertext Transfer Protocol Secure) HTTPS als Kommunikationsmittel.

Da REST als äußerst benutzerfreundlich für Datenabfragen und -übertragungen gilt, ist die Verwendung von REST bei Software-Entwicklern sehr beliebt.[9]

[6] vgl. online: HEINEN Elektronik GmbH, 2023 (01.07.2024)
[7] vgl. online: HEINEN Elektronik GmbH, 2023 (01.07.2024)
[8] vgl. online: HEINEN Elektronik GmbH, 2023 (01.07.2024)
[9] vgl. online: KASUBKE, 2020 (01.07.2024)

Eine Vielzahl an Systemen bietet bereits eine REST-API und stellt darüber eine standardisierte Möglichkeit bereit, Daten und Funktionalität mit anderer Software auszutauschen.[10]

Diese Schnittstelle kann für eine Vielzahl von Anwendungen und Servern eingesetzt werden, die das HTTP-Protokoll verwenden. Beispiele dazu sind: Webanwendungen, Cloud-Dienste, mobile Anwendungen und E-Commerce.

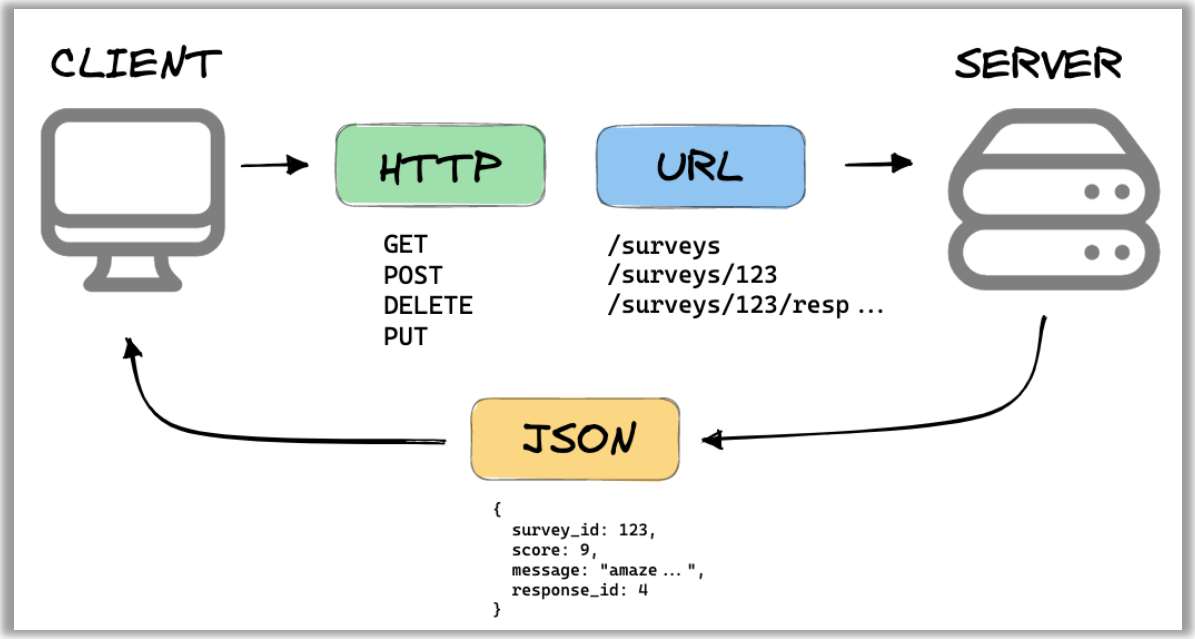


Abbildung 3 Ein Beispiel zur REST-Schnittstelle[11]
(mannhowie.com, 10.07.2024)

2.5.1.2 Aufbau

Eine REST-API Anfrage wird beispielweise für eine Auflistung aller Kundeninformationen von einem Server wie folgt aufgebaut:

```
GET https://api.example.com/kunden
```

Abbildung 4 Der Aufbau einer REST-API-Anfrage

[10] vgl. online: KASUBKE, 2020 (01.07.2024)
[11] online: REST APIs Explained - 4 Components, 2024 (01.07.2024)

Bestandteil	Erklärung
GET	Die HTTP-Operation, diese kann GET (Abrufen), POST (Erstellen), PUT (Aktualisieren) oder DELETE (Löschen) sein.
https://api.examle.com	Die Basis-URL der API, zu der eine Verbindung hergestellt wird
kunden	Der Endpunkt der API, der auf die Datenressource verweist.

Tabelle 1 Die Bestandteile einer REST-API-Anfrage

Die Antwort auf die o.g. Anfrage könnte wie folgt aussehen, falls der Server erreichbar war und die Anfrage korrekt aufgebaut wurde:

Abbildung 5 Der Aufbau einer REST-API-Antwort

Bestandteil	Erklärung	Im Beispiel
HTTP/__	Das für die Übertragung verwendete Protokoll mit Angabe der Version.	HTTP/1.1
Code	Statuscode der Übertragung	200 OK
Content-Type	Klassifikation der Daten in der Antwort	application/json
JSON-Text	Der Inhalt der Antwort vom Server mit allen angeforderten Objekten.	Die Kunden mit all ihren Eigenschaften

Tabelle 2 Die Bestandteile einer REST-API-Antwort

2.5.1.3 Vor- und Nachteile

Vorteile	Nachteile
• Gute Skalierbarkeit durch das HTTP-Protokoll, welches von vielen Servern und Clients unterstützt wird. • Zustandslos, da jede Anfrage von Clients alle Informationen enthält, die zur Verarbeitung der Anfrage erforderlich sind. • Einfache Wartung	• Begrenzte Funktionalität, da REST auf die CRUD-Operationen (Create, Read, Update, Delete) beschränkt ist. • Keine Standardisierung für die Gestaltung von REST-APIs • Overfetching und Underfetching durch das Abrufen von mehr bzw. weniger Daten als benötigt, da die API-Ressourcen vordefiniert sind.

Tabelle 3 Die Vor- und Nachteile einer REST-API-Schnittstelle

2.5.2 SOAP

2.5.2.1 Definition

Simple Object Access Protocol (SOAP) ist ein Nachrichtenprotokoll, das die Kommunikation zwischen verteilten Komponenten einer Anwendung ermöglicht. SOAP kann über verschiedene Standardprotokolle wie das webbezogene HTTP übertragen werden. Außerdem ist SOAP ein leichtgewichtiges Protokoll, das zur Erstellung von Web-APIs verwendet wird, normalerweise mit der Extensible Markup Language (XML). Es unterstützt eine breite Palette von Kommunikationsprotokollen im Internet,

HTTP, Simple Mail Transfer Protocol (SMTP) und Transmission Control Protocol (TCP).[12]

Dieses Protokoll wurde für Anwendungen mit verschiedenen Programmiersprachen als Sekundärsprache entwickelt, damit diese Anwendungen über das Internet miteinander kommunizieren können. Das Protokoll ist unabhängig und flexibel, so dass Entwickler SOAP-Programmierschnittstellen in verschiedenen Sprachen schreiben und gleichzeitig Merkmale und Funktionen hinzufügen können.[13]

2.5.2.2 Aufbau

Ein Beispiel über die Implementierung einer Anfrage mittels dieser Technologie könnte wie folgt gebildet werden:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:web="http://www.example.com/webservice">
    <soapenv:Header/>
    <soapenv:Body>
        <web:GetBookInfo>
            <web:BookID>123</web:BookID>
        </web:GetBookInfo>
    </soapenv:Body>
</soapenv:Envelope>
```

Abbildung 6 Der Aufbau einer SOAP-Anfrage

Bestandteil	Erklärung	Im Beispiel
<soapenv:Envelope>	Das ist das Wurzelelement, das die gesamte SOAP-Nachricht umgibt. Es definiert den XML-Namespace für SOAP.	
<soapenv:Header>	Es enthält optionale Header-Informationen	
<soapenv:Body>	Es enthält den Hauptteil der SOAP-Anfrage.	
<web:GetBookInfo>	Das ist das Element, das die spezifische SOAP-Aktion oder den gewünschten Service angibt.	

[12] vgl. online: ComputerWeekly.de, 2024 (01.07.2024)
[13] vgl. online: ComputerWeekly.de, 2024 (01.07.2024)

<web:BookID>	Das ist das Argument für die Anfrage, für die Sie Informationen abrufen möchten.	Spezifische Buch-ID 123

Tabelle 4 Die Bestandteile einer SOAP-Anfrage

Die Antwort auf die o.g. Anfrage lautet:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:web="http://www.example.com/webservice">
    <soapenv:Header/>
    <soapenv:Body>
        <web:GetBookInfoResponse>
            <web:Book>
                <web:Title>Buch 1</web:Title>
                <web:Author>Autor 1</web:Author>
                <web:PublicationYear>2020</web:PublicationYear>
            </web:Book>
        </web:GetBookInfoResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Abbildung 7 Der Aufbau einer SOAP-Antwort

2.5.2.3 Vor- und Nachteile

Vorteile	Nachteile
• Plattform- und Betriebssystemunabhängigkeit • Über verschiedene Netzwerk- und Security-Geräte übertragbar	• Keine Übermittlung der Daten per Referenz • Langsame Verarbeitung durch die Größe der Pakete

Tabelle 5 Die Vor- und Nachteile einer SOAP-Schnittstelle

2.5.3 GraphQL

2.5.3.1 Definition

GraphQL ist eine Abfragesprache und serverseitige Umgebung für APIs, die es ermöglicht, genau die benötigten Daten an die Clientanwendungen zu liefern.

Diese Technologie verleiht APIs eine höhere Geschwindigkeit, größere Flexibilität und erhöht die Benutzerfreundlichkeit für Entwickler erheblich. Sie kann sogar in einer integrierten Entwicklungsumgebung (IDE) namens GraphiQL genutzt werden. Im Gegensatz zur herkömmlichen REST-Architektur erlaubt es GraphQL Entwicklern,

Abfragen zu formulieren, die in einem einzigen API-Aufruf Daten aus verschiedenen Quellen gleichzeitig abrufen können.

Darüber hinaus bietet GraphQL API-Anbietern die Flexibilität, Felder hinzuzufügen oder zu entfernen, ohne dabei bestehende Abfragen zu beeinträchtigen. Entwickler haben die Freiheit, APIs nach ihren Vorlieben zu gestalten und die GraphQL-Spezifikation stellt sicher, dass die API konsistent und vorhersehbar für die Clientanwendungen funktioniert.[14]

2.5.3.2 Aufbau

Abbildung 8 Der Aufbau einer GraphQL-Anfrage

Bestandteil	Erklärung
Query	Schlüsselwort, das anzeigt, dass Daten abgerufen werden.
Books	Die Art der abgefragten Daten
Title, author und publicationYear	Die spezifischen Informationen der abgefragten Daten

Tabelle 6 Die Bestandteile einer GraphQL-Anfrage

[14] vgl. online: GraphQL erklärt, 2024 (02.07.2024)

Die Antwort auf die o.g. Anfrage könnte wie folgt aussehen:

Abbildung 9 Der Aufbau einer GraphQL-Antwort

2.5.3.3 Vor- und Nachteile

Vorteile	Nachteile
• Eine Weiterentwicklung von Anwendungs-API ohne Beeinträchtigung bestehender Systeme • Keine spezifische Anwendungsarchitektur, da es auf einer vorhandenen REST-API installiert werden kann. • Verringertes Risiko einer Fehlkommunikation zwischen Client und Server mithilfe definierter Datentypen • Kein Overfetching oder Underfetching	• Weiterer Lernaufwand für Entwickler mit REST-API Kenntnissen. • Komplexes Caching, da GraphQL keine standardisierten HTTP-Caching-Methoden unterstützt und stattdessen aufwendige Konfiguration und Verwaltung von clientseitigem Caching erfordert. • Weiterer Aufwand für API-Maintainer zum Schreiben von verwaltbaren GraphQL-Schemata.

Tabelle 7 Die Vor- und Nachteile einer GraphQL-Schnittstelle[15]

[15] vgl. online: ESEME, 2022 (08.08.2024)

Viele Unternehmen und Organisationen haben GraphQL in ihren Technologie-Stack integriert. Hier sind einige bekannte Unternehmen, die GraphQL verwenden: Facebook, GitHub, Twitter, Shopify, Airbnb und Uber etc.[16]

Folgende Abbildung zeigt den Unterschied zwischen REST und GraphQL:

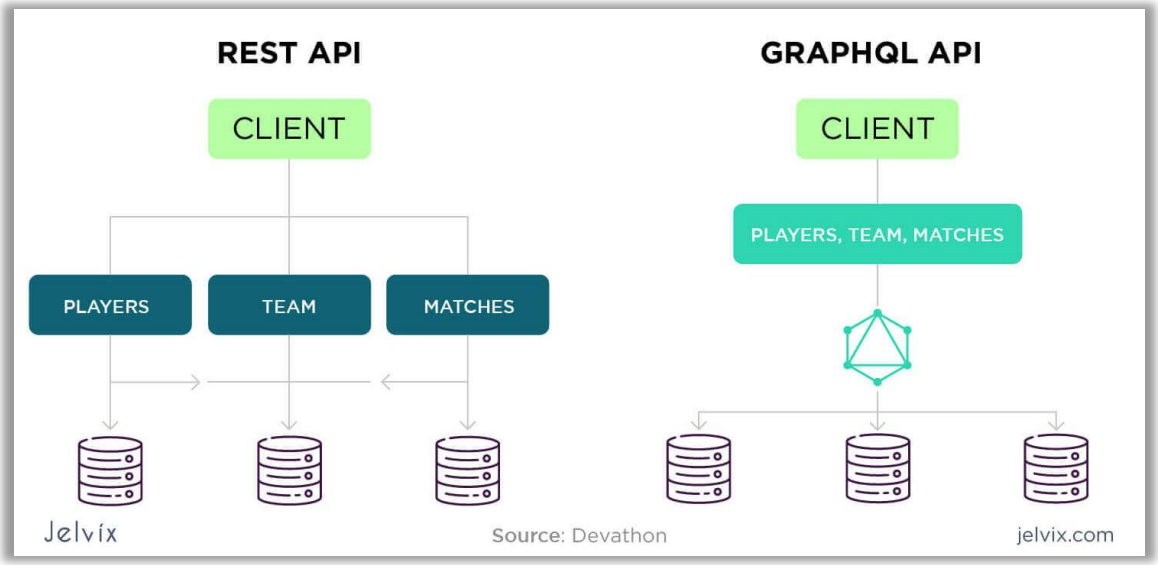


Abbildung 10 Der Unterschied zwischen REST-API und GraphQL-API[17] (jelvix.com, 02.07.2024)

Im Gegensatz zur REST-API bietet GraphQL eine höhere Flexibilität, da Entwickler clientseitig bestimmen können, welche Daten sie gleichzeitig mit nur einem Aufruf aus verschiedenen Quellen anfordern möchten. Im Gegensatz dazu ist bei REST-API die Datenstruktur serverseitig vordefiniert und es erfordert mehrere separate Aufrufe, um dieselben Daten abzufragen.

2.5.3.4 Unterschied zu SQL

GraphQL und SQL sind zwei unterschiedliche Ansätze zur Datenabfrage und -verwaltung, die jeweils spezifische Stärken haben. SQL ist die traditionelle Sprache für relationale Datenbanken, bei der Daten in Tabellen gespeichert und Abfragen über Joins und Aggregationen ausgeführt werden. SQL neigt dazu, mehr Daten abzurufen als nötig, es sei denn, Abfragen sind spezifisch gefiltert, und arbeitet oft mit mehreren API-Endpunkten. GraphQL hingegen bietet eine flexiblere Methode, indem es Clients ermöglicht, genau die benötigten Datenfelder in einer einzigen Abfrage abzurufen. Es vermeidet unnötige Datenübertragungen und unterstützt verschachtelte Datenstrukturen sowie Echtzeit-Updates, was es besonders geeignet für moderne

[16] vgl. online: Matthias Zeis, 2022 (04.08.2024)
[17] online: Jelvix, 2020 (02.07.2024)

Anwendungen macht. GraphQL verwendet einen einzigen Endpunkt für alle Abfragen, was die API-Verwaltung vereinfacht.[18]

GraphQL eignet sich ideal für Echtzeit-Anwendungen, mobile Apps und Mikroservices, während SQL besser für datenintensive Anwendungen, Legacy-Systeme und komplexe Berichte geeignet ist. Die Wahl zwischen GraphQL und SQL hängt von den spezifischen Anforderungen des jeweiligen Projekts ab.

[18] vgl. online: mobileLIVE, 2023 (31.07.2024)

3 Stand der Forschung und Technik

3.1 Aktuelle Trends und Best Practices in der Schnittstellenentwicklung

Die moderne Schnittstellenentwicklung wird stark durch standardisierte Protokolle wie REST und GraphQL geprägt, weil sie flexible und effiziente Datenabfragen von einem Server ermöglichen. API-Gateways spielen dabei eine wichtige Rolle, indem sie Funktionen wie Authentifizierung und Lastverteilung übernehmen.

Automatisierung und API-Management-Plattformen sind ebenfalls bedeutend, da sie die Verwaltung, Dokumentation und Überwachung von Schnittstellen erleichtern. Der API-Design-First-Ansatz fördert eine klare Definition der Schnittstellen, bevor die Implementierung beginnt, und Tools wie Swagger unterstützen diese Vorgehensweise durch automatische Dokumentation.[19]

Die Sicherheit der Schnittstellen wird durch den Einsatz von Technologien wie O-Auth2 und JWT gewährleistet, die den Zugriff auf Schnittstellen schützen und die Datenintegrität sicherstellen. Zusammengefasst zielen diese Trends darauf ab, die Schnittstellenentwicklung effizienter, sicherer und besser wartbar zu gestalten.

Eine gut strukturierte Schnittstelle sollte modular entwickelt werden, um Änderungen und Erweiterungen einfach umsetzen zu können, ohne die Gesamtstruktur zu beeinträchtigen. Zudem spielt das Monitoring und Logging eine große Rolle bei einer zuverlässigen Schnittstelle, denn nur mithilfe einer detaillierten Überwachung können Probleme und Ausfälle schnell identifiziert und behoben werden.[20]

Die Verwaltung von Feldzuordnungen sollte einfach und für Anwender ohne tiefgehende technische Vorkenntnisse möglich sein. Da die Zuordnungen vor der Abnahme eines Projekts sehr oft angepasst werden könnten, insbesondere wenn die zu übertragenden Informationen umfangreich sind, ist eine dynamische und flexible Konfiguration während der Entwicklungs- und Testphase erforderlich. Diese sollte so gestaltet sein, dass die Zuordnung der Felder zwischen zwei oder mehreren Objekten den Kundenerwartungen genau entspricht.

Eine derartige Konfigurationsmöglichkeit bietet die Software Altova MapForce. Altova MapForce ist ein preisgekröntes grafisches Tool für das Daten-Mapping und die Integration verschiedenster Datentypen. Es ermöglicht eine sofortige Konvertierung

[19] vgl. online: Astera, 2023 (02.07.2024)
[20] vgl. online: Schnittstellen-Programmierung: Einfache Grundlagen, die begeistern!, 2024 (02.07.2024)

Ihrer Daten und bietet zahlreiche Optionen zur Automatisierung wiederkehrender Konvertierungsprozesse.[21]

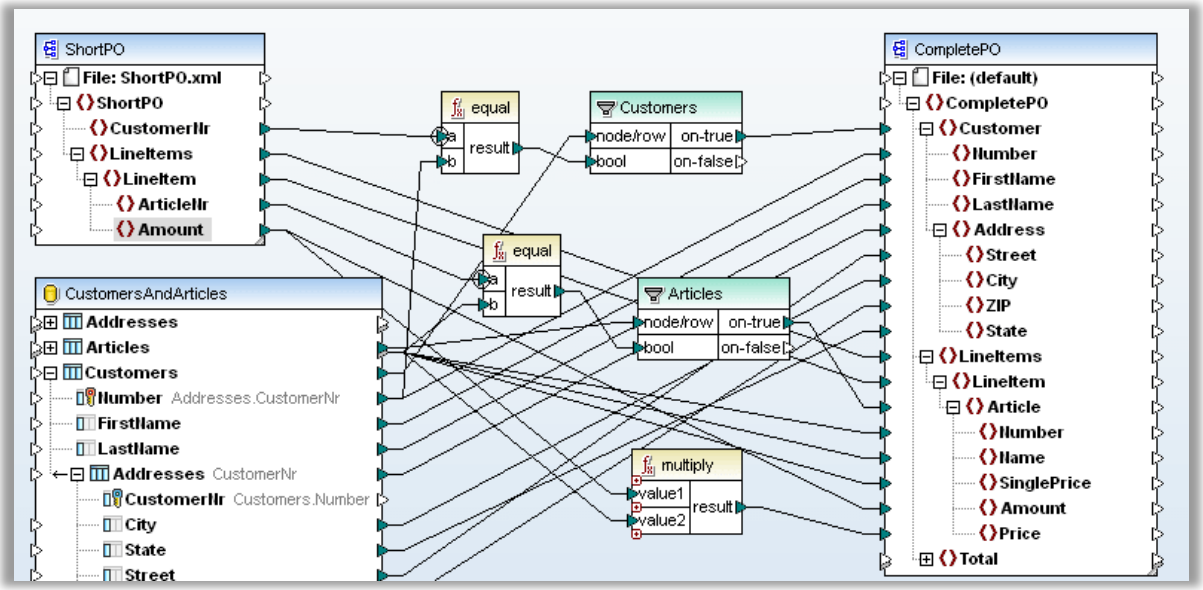


Abbildung 11 Altova MapForce Beispiel[22]
(componentsource.com, 03.07.2024)

Jedoch wird diese Software bei uns intern nicht verwendet, da sie nicht zu unserer internen Struktur passt. Die Konfiguration der Schnittstelle mit Altova MapForce würde aufgrund des hohen Konfigurationsaufwands erheblich viel Zeit in Anspruch nehmen. Zudem ist die Software mit unstrukturierten Schnittstellentypen, wie Dateischnittstellen, TCP-String-Übertragungen und Byte-Arrays, inkompatibel. Ein weiterer Nachteil ist, dass die Software lizenzpflichtig und damit sehr teuer ist.

3.2 Bestehende Lösungen und deren Herausforderungen

Die aktuellen Lösungen zur Verwaltung von Schnittstellen erfordern häufig manuelle Anpassungen im Quellcode, was nicht nur zeitaufwendig ist, sondern auch ein erhöhtes Risiko für Fehler birgt.

Ein weiteres Problem besteht darin, dass bei jedem neuen Kunden die Schnittstellenstruktur entweder neu entwickelt oder aus vorherigen Projekten kopiert wird. Dies führt häufig zur Übernahme spezifischer Anforderungen aus alten Projekten in neue Kontexte, was die Gefahr von Inkompatibilitäten und zusätzlichen Anpassungen erhöht. Außerdem wird bei der Übertragung von Schnittstellen aus alten in neue Kundenprojekte die Performance gefährdet, weil dadurch unnötiger Overhead entsteht, der für die Schnittstelle des neuen Kundenprojekts nicht erforderlich ist.

[21] vgl. online: Datenmapping-Tools: MapForce, 2024 (03.07.2024)
[22] online: ComponentSource, 2024 (03.07.2024)

Zudem sind die bestehenden Systeme oft nicht standardisiert und mangelhaft dokumentiert, was die Wiederverwendbarkeit und Interoperabilität erheblich einschränkt.

Die Komplexität der Konfiguration und die Abhängigkeit von Expertenwissen erschweren die Pflege und Weiterentwicklung der Schnittstellen, was zu verlängerten Entwicklungszyklen und höheren Kosten führt. Ferner fehlt es den meisten Lösungen an Benutzerfreundlichkeit, wodurch die Verwaltung der Schnittstellen für Nutzer ohne tiefgehende technische Kenntnisse eine Herausforderung darstellt.

Aus diesen Vorgehensweisen resultieren insgesamt in einer hohen Fehleranfälligkeit und ineffizienten Projektabläufen.

Um solche Fehler zu vermeiden, soll für das Unternehmen eine modulare und generische Standard-Schnittstelle entwickelt werden, die nur die essenziellen Komponenten für eine lauffähige Schnittstelle beinhaltet, mit allen Schnittstellentypen kompatibel ist und in Kunden-Projekte einfach erweitert werden kann. Für die Administration der Schnittstelle sollte eine benutzerfreundliche GUI entwickelt werden, damit Nutzer, Tester oder auch Projektleiter die Schnittstelle ohne technische Vorkenntnisse konfigurieren können, beispielsweise um die Felder zwischen dem internen und externen System zuzuordnen.

3.3 Anforderungen an eine benutzerfreundliche GUI

Ben Shneiderman, geboren am 21. August 1947, ist ein renommierter US-amerikanischer Informatiker und Professor am Human-Computer Interaction Lab der University of Maryland. Seine Forschungen und Ideen stehen in einer Reihe mit anderen einflussreichen Denkern im Bereich Design wie Don Norman und Jakob Nielsen. In seinem bekannten Buch „Designing the User Interface: Strategies for Effective Human-Computer Interaction“ stellt Shneiderman seine acht goldenen Regeln für ein erfolgreiches Interface-Design vor:[23]

1. **Konsistenz sicherstellen:** Es sollte darauf geachtet werden, dass vertraute Symbole, Farben, Menühierarchien und Benutzerabläufe verwendet werden. Eine Standardisierung der Informationsübermittlung ist ebenfalls wichtig, damit Benutzer ihr Wissen anwenden können. Ziel ist es, den Benutzern zu ermöglichen, sich in der digitalen Umgebung leichter zurechtzufinden.[24]
2. **Unterstützung von Vielnutzern durch Tastenkombination:** Tastenkombinationen für häufig genutzte Funktionen sollen angeboten werden. Dies ermöglicht erfahrenen Benutzern, Aufgaben schneller und effizienter zu erledigen.[25]

[23] vgl. online: The Interaction Design Foundation, 2024 (03.07.2024)
[24] vgl. online: The Interaction Design Foundation, 2024 (03.07.2024)
[25] vgl. online: The Interaction Design Foundation, 2024 (03.07.2024)

3. **Informatives Feedback bieten:** Benutzer sollten jederzeit wissen, wo sie sich befinden und was gerade geschieht. Dafür muss klar lesbar und zeitnahes Feedback gegeben werden, beispielsweise durch Fortschrittsanzeigen bei mehrseitigen Prozessen.[26]
4. **Dialog zum Abschluss führen:** Nach Aktionen sind klare Rückmeldungen, wie Bestätigungsnachrichten bei einem Kauf, notwendig. So wird vermieden, dass Benutzer im Unklaren über den Status ihrer Aktionen bleiben.[27]
5. **Einfache Fehlerbehandlung anbieten:** Systeme sollten so gestaltet sein, dass sie auch von unerfahrenen Nutzern problemlos bedient werden können. Dabei ist es wichtig, klare Anweisungen zur Fehlerbehebung bereitzustellen. Beispielsweise ist das Hervorheben fehlender Eingaben in Formularen eine nützliche Maßnahme.[28]
6. **Einfache Umkehrung von Aktionen ermöglichen:** Es müssen Optionen zur Rückgängigmachung von Aktionen bereitgestellt werden, um Benutzerängste zu reduzieren. Diese Möglichkeiten sollten an verschiedenen Stellen der Nutzerinteraktion verfügbar sein.[29]
7. **Interne Kontrollüberzeugung unterstützten:** Benutzern soll die Möglichkeit gegeben werden, selbstständig Aktionen auszuführen und Kontrolle über das System zu behalten. Dabei ist es wichtig, dass das System sich erwartungsgemäß verhält, um Vertrauen aufzubauen und eine positive Nutzererfahrung zu gewährleisten.[30]
8. **Belastung des Kurzzeitgedächtnisses reduzieren:** Benutzeroberflächen sollten so gestaltet sein, dass sie leicht verständlich sind und das Wiedererkennen von Informationen erleichtern, anstatt den Nutzer zum aktiven Erinnern zu zwingen. Eine klare und gut strukturierte Darstellung der Informationen ist entscheidend, um die geistige Belastung der Nutzer so gering wie möglich zu halten.[31]

[26] vgl. online: The Interaction Design Foundation, 2024 (03.07.2024)
[27] vgl. online: The Interaction Design Foundation, 2024 (03.07.2024)
[28] vgl. online: The Interaction Design Foundation, 2024 (03.07.2024)
[29] vgl. online: The Interaction Design Foundation, 2024 (03.07.2024)
[30] vgl. online: The Interaction Design Foundation, 2024 (03.07.2024)
[31] vgl. online: The Interaction Design Foundation, 2024 (03.07.2024)

4 Methodik

4.1 Literaturrecherche

Die Literaturrecherche stellt einen wesentlichen Teil dieser Bachelorarbeit dar und fokussiert sich auf die Analyse relevanter wissenschaftlicher Artikel, Fachbücher und Online-Ressourcen. Sie umfasst eine eingehende Untersuchung der aktuellen Forschung und praktischen Anwendungen im Bereich Schnittstellenentwicklung und des GUI-Designs. Besondere Aufmerksamkeit wird auf bewährte Verfahren gelegt, um benutzerfreundliche Schnittstellen für komplexe Systeme zu schaffen. Dabei werden unterschiedliche Faktoren in Betracht gezogen, wie zum Beispiel Usability-Prinzipien, Informationsdesign, Interaktionsmuster und Fallbeispiele erfolgreicher Umsetzungen.

Diese ausführliche Recherche in der Literatur schafft eine solide theoretische Grundlage, die bei der Entwicklung und Umsetzung einer wirksamen Lösung zur Vereinfachung der Verwaltung und zur Verallgemeinerung von Unternehmenssystem-Schnittstellen helfen wird.

4.2 Softwareentwicklungswerkzeuge und -technologien

Für die Entwicklung des GUI-Mappers und der modular generalisierten Schnittstelle wurden spezifische Softwareentwicklungswerkzeuge und -technologien eingesetzt, um eine effiziente und effektive Implementierung sicherzustellen. Im Folgenden werden die wesentlichen Tools und Technologien beschrieben, die im Rahmen dieses Projekts verwendet wurden:

- **Java 1.8**
  Für die Entwicklung der generalisierten Schnittstelle sowie die Erstellung der GUI wurde Java in der Version 1.8 eingesetzt werden. Java 1.8, auch bekannt als Java 8, bietet umfangreiche Funktionalitäten, darunter Lambdas, das Stream-API und zahlreiche weitere Features, die die Entwicklung moderner und effizienter Anwendungen unterstützen.[32]
  Diese Version von Java wurde gewählt, damit die Schnittstelle mit unserer Programmiersprache vom WMS zusammenpasst und es keine Konflikte gibt.
- **JavaFX**
  JavaFX ist eine Java-Plattform, die sich auf die Entwicklung grafischer und multimedialer Anwendungen spezialisiert. Sie ermöglicht die Erstellung interaktiver Benutzeroberflächen und eignet sich sowohl für Desktop- als auch für mobile Anwendungen. JavaFX bietet eine moderne, effiziente und umfassend

[32] vgl. online: Informationen zu Java 8, 2024 (04.07.2024)

ausgestattete Toolkit-Umgebung zur Entwicklung ansprechender Client-Anwendungen.[33]

Im WMS wurden bisher sämtliche grafischen Oberflächen und Werkzeuge mit JavaFX entwickelt. Dadurch ist JavaFX im Laufe der Zeit tief in das System integriert worden und es wurden mehrere selbst entwickelte Hilfsmodule erstellt, die den Entwicklern einen Großteil der Zusatzarbeit abnehmen. Aus diesem Grund sollte auch bei der Administration der Schnittstellen von diesen Modulen künftig profitiert werden.

- **Eclipse IDE 4.29.0**

  Für die Entwicklung der modularen Schnittstelle kam die Eclipse IDE in der Version 4.29.0 zum Einsatz. Diese integrierte Entwicklungsumgebung, (auf Englisch, Integrated Development Environment - IDE) ist ein Open-Source-Programmierwerkzeug, das für die Programmiersprache Java begann, aber mittlerweile auch viele andere Entwicklungsaufgaben übernimmt. Es bietet eine plattformunabhängige Umgebung für verschiedene Programmiersprachen und Metasprachen.[34]

  Es wurde mit Eclipse gearbeitet, da es bei der iFD aufgrund der vielen selbst entwickelten Plugins die standardmäßige Entwicklungsumgebung für unsere Softwareentwicklung ist. Zu den relevanten Plugins gehört die Erweiterung des SceneBuilder, damit die selbst entwickelten GUI-Komponenten im SceneBuilder akzeptiert und eingelesen werden.

- **Apache Ant 2.0**

  Apache Ant ist ein in Java geschriebenes Programm zum automatisierten Erzeugen von ausführbaren Computerprogrammen aus Quelltexten. Es erfüllt die automatisierte Generierung von installierbaren Software-Paketen aus vorhandenen Quelltexten, Bibliotheken und anderen Dateien. Im Wesentlichen wird Ant für die Erstellung und das Deployment von Java-Projekten eingesetzt.[35]

  Seit langer Zeit verwendet die iFD Apache Ant als Build-Tool für das WMS. Es existieren auch eigene Build-Skripte, die auf diesem Tool basieren. Dementsprechend ist es wichtig, Apache Ant als Build-Tool beizubehalten.

- **Hibernate 5.4.21**

  Hibernate ist ein Open-Source-Persistenz- und ORM- Framework (Object-Relational Mapping) für Java-Anwendungen. Es wurde entwickelt, um die Lücke zwischen objektorientierter Programmierung und relationalen Datenbanken zu überbrücken. Durch Hibernate wird die Speicherung, der Zugriff und

[33] vgl. online: Wikipedia, 2024c (04.07.2024)
[34] vgl. online: Wikipedia, 2024b (04.07.2024)
[35] vgl. online: Wikipedia, 2024a (04.07.2024)

die Verwaltung von Daten in einer relationalen Datenbank mit Hilfe von Java-Objekten erheblich erleichtert. Hibernate verfügt über ein leistungsfähiges Caching-System, welches die Leistung verbessert, indem es die Anzahl der Datenbankzugriffe reduziert.[36]

In der iFD ist Hibernate das Standard-Framework im Bereich WMS, aufgrund dessen bestand stets die Anforderung, Hibernate weiterhin zu verwenden.

- **SQL Server Management Studio 19.3.4.0**

  Bei der Generalisierung der Schnittstelle wurde für die Verwaltung, Überwachung und Analyse der Ergebnisse in der Datenbank das SQL Server Management Studio in Version 19.3.4.0 verwendet.

  Das SQL Server Management Studio (SSMS) ist eine umfassende integrierte Entwicklungsumgebung von Microsoft und wurde speziell für die Verwaltung von SQL Server-Datenbanken entwickelt. Es bietet leistungsstarke Werkzeuge zur Konfiguration, Überwachung und Diagnose von Datenbanken sowie zur Ausführung und Optimierung von Abfragen.[37]

- **SoapUI 5.7.2**

  SoapUI ist das weltweit am häufigsten verwendete automatisierte Testwerkzeug für SOAP- und REST-APIs. Mit SoapUI können Entwickler Schnittstellen testen und die Effizienz der gesamten Teststrategie deutlich verbessern. Dabei können Webdienste aufgerufen und simuliert werden. SoapUI ermöglicht unter anderem auch Last- und Konformitätstests.[38]

- **Git**

  Git ist ein Open-Source-System zur Versionsverwaltung von Projekten, das es ermöglicht, während der Entwicklung zu früheren Versionen zurückzuspringen, um Änderungen zu überprüfen. Es kann über die Kommandozeile oder durch spezielle Git-Clients bedient werden, die eine grafische Oberfläche bieten. Ein zentraler Aspekt von Git ist die Möglichkeit, Änderungen über Repositories zu teilen und gemeinsam an Projekten zu arbeiten. Durch die Verwendung von Branches kann nachverfolgt werden, an welchen Features gearbeitet wird, bevor sie in den Hauptzweig (Master) übernommen werden. Bei Bedarf kann einfach auf frühere, stabile Versionen zurückgegriffen werden. Git wurde von Linus Torvalds, dem Entwickler des Linux-Betriebssystemkernels, initiiert.[39]

[36] vgl. online: Deinhard, 2024 (04.07.2024)
[37] vgl. online: Markingmyname, 2024 (04.07.2024)
[38] vgl. online: Das weltweit beliebteste API-Testtool | SoapUI, 2024 (04.07.2024)
[39] vgl. online: Git und Github: Unterschiede einfach erklärt, 2024 (08.07.2024)

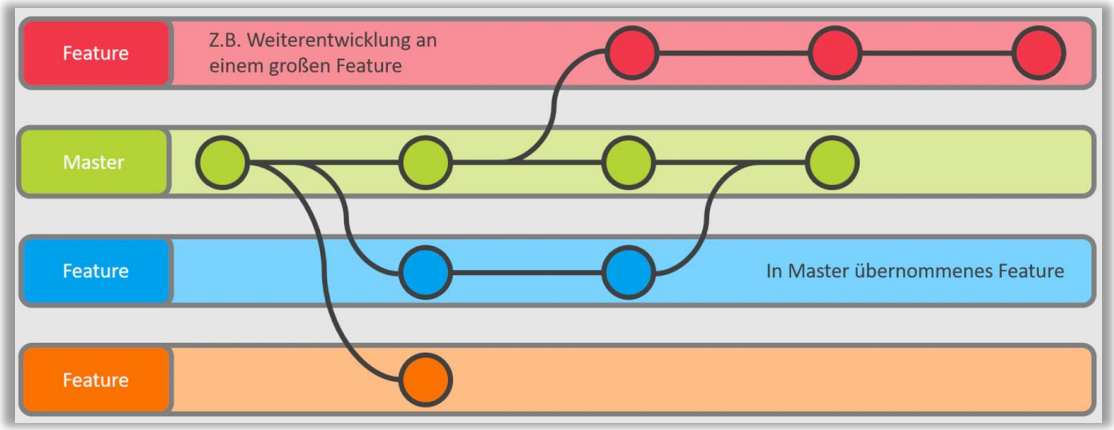


Abbildung 12 Git Parallelentwicklung von Features[40]
(praxistipps.chip.de, 08.07.2024)

Während der gesamten Entwicklung wurde der Quellcode sowohl für die allgemeine Schnittstelle als auch für die GUI zur Verwaltung von Schnittstellen in GitLab gesichert und dokumentiert. Zunächst wurde in einem separaten Branch gearbeitet, und erst nach erfolgreichen Tests erfolgte der Merge in den Master-Branch.

- **Spring-Framework zur XML-Schema-basierten Konfiguration**
  Die XML-Schema-basierte Konfiguration im Spring-Framework ermöglicht eine strukturierte und erweiterbare Art der Konfiguration von Beans im Kontext der Inversion of Control (IoC) Container.
  Beans sind Java-Objekte, die als Bestandteil einer Spring-Anwendung erstellt werden und vom Spring IoC-Container verwaltet werden. Der Container ist verantwortlich für das Instanziieren der Beans, das Konfigurieren ihrer Abhängigkeiten (z. B. durch Dependency Injection) und das Verwalten ihres Lebenszyklus.[41]
  Diese Technik wurde in Spring 2.0 eingeführt und bietet eine klarere und prägnantere Alternative zur älteren Document-Type-Definition (DTD)-basierten Konfiguration. Mit der XML-Schema-basierten Konfiguration können Spring-Beans in einer bestimmten Anwendung definiert und konfiguriert werden. Hierbei werden Konfigurationsdetails direkt in den Java Klassen angegeben, was die Lesbarkeit und Wartbarkeit des Codes verbessern kann.[42]
  Das Spring-Framework wurde in dieser Arbeit eingesetzt, damit die Parameter von Schnittstellen im WMS-Server konfiguriert werden.

[40] online: Git und Github: Unterschiede einfach erklärt, 2024 (08.07.2024)
[41] vgl. online: Container Overview :: Spring Framework, 2024 (05.08.2024)
[42] vgl. online: 40. XML Schema-based configuration, 2024 (08.07.2024)

In folgendem Beispiel wird die Funktionalität des Spring-Frameworks zur XML-Schema-basierten Konfiguration anhand der Klassen „*PersistenceContext-Type*“ und „*Client*“ veranschaulicht:

```
package example;

public enum PersistenceContextType {
    TRANSACTION,
    EXTENDED
}
```

Abbildung 13 Spring-Framework Beispiel; Enum „*PersistenceContextType*“[43] (docs.spring.io, 08.07.2024)

```
package example;

import javax.persistence.PersistenceContextType;

public class Client {

    private PersistenceContextType persistenceContextType;

    public void setPersistenceContextType(final PersistenceContextType type) {
        this.persistenceContextType = type;
    }
}
```

Abbildung 14 Spring-Framework Beispiel; Klasse „*Client*“ [44] (docs.spring.io, 08.07.2024)

Nun sollte beispielsweise die Klasse „*Client*“ mit dem „*PersistenceContext-Type*“ „*TRANSACTION*“ als Eigenschaft initialisiert werden. Die Konfiguration würde mit Spring-Framework wie folgt aussehen:

```
<bean class="example.Client">
    <property name="persistenceContextType" value="TRANSACTION" />
</bean>
```

Abbildung 15 Spring-Framework Beispiel; Initialisierung der Klasse „*Client*“ mit einer Bean-Definition[45] (docs.spring.io, 08.07.2024)

[43] online: 40. XML Schema-based configuration, 2024 (08.07.2024)
[44] online: 40. XML Schema-based configuration, 2024 (08.07.2024)
[45] online: 40. XML Schema-based configuration, 2024 (08.07.2024)

5 Generalisierung der Schnittstelle

5.1 Erklärung

Die Generalisierung der Schnittstelle im WMS ist ein zentraler Aspekt dieser Arbeit. In diesem Abschnitt wird ausführlich beschrieben, wie die generische Schnittstelle im WMS entwickelt und eingesetzt wurde, um ein einheitliches Schnittstellen-System zu erschaffen und somit die Entwicklungsstunden eines Projekts zu reduzieren.

5.2 Anforderung an eine generische Schnittstelle

Eine modulare generische Standard-Schnittstelle im WMS in der iFD soll hauptsächlich dazu beitragen, den zur Abwicklung eines Projekts erforderlichen Aufwand möglichst zu reduzieren und dadurch den Ertrag zu maximieren.

Die Standard-Schnittstelle im WMS soll dabei den Großteil einer wiederkehrenden Schnittstellenstruktur beinhalten und auf überflüssigen Overhead verzichten, sodass nur die wesentlichen Funktionen integriert werden.

Ein weiterer wesentlicher Aspekt einer generischen Standardschnittstelle ist die Fähigkeit, eine Zuordnung zwischen Telegramm-Kategorien und Konvertern zu erstellen, die die Kommunikation zwischen dem Host-System und dem WMS ermöglichen. Ein Telegramm entspricht dabei einer Information, die von einem System zum anderen übertragen wird. Jedes Telegramm wird kategorisiert, um es im nächsten Schritt durch den entsprechenden Konverter verarbeiten zu lassen.

Je nach empfangenem oder versendetem Datenpaket soll der entsprechende Konverter für das Daten-Mapping verantwortlich sein. Diese Konverter sollen über das Spring-Framework im Server einfach registriert werden können, was die Flexibilität und Anpassungsfähigkeit der Schnittstelle weiter erhöht.

Es sollen Standardfunktionen bereitgestellt werden, die bei jedem Projekt einheitlich entwickelt werden, um Fehler und Bugs in Kundenprojekten zu vermeiden. Ein Beispiel hierfür ist eine Funktion zur kontinuierlichen Überwachung von Datenbanken.

Eine generische Schnittstelle soll darüber hinaus in der Lage sein, in jedem Kundenprojekt leicht erweiterbar zu sein, um spezifische Kundenanforderungen zielorientiert umsetzen zu können.

Ferner sollen die ein- und ausgehenden Datenpakete vollumfänglich mit Kategorie, Zeitstempel und Inhalt geloggt werden, um ein einfaches Monitoring zu gewährleisten und potenzielle Probleme schneller zu erkennen.

Zusammengefasst soll eine generische Standard-Schnittstelle das Grundgerüst jeder Schnittstelle bilden. Zudem soll diese generalisierte Schnittstelle in unser Basisprodukt integriert werden, sodass sie bei neuen Projekten standardmäßig vorhanden ist.

5.3 Evaluationsmethode

Für die Ausführung dieser Arbeit wurde die qualitative Forschungsmethode angewendet, um ein tiefes Verständnis für die Anforderungen und Funktionsweisen einer generischen Schnittstelle zu erlangen. Im Rahmen dieser Methode fanden mehrere Expertengespräche statt, um ein umfassendes Verständnis für die Anforderungen und Arbeitsweisen einer generischen Schnittstelle sowie deren wesentliche Funktionen im System zu erlangen.

Basierend auf den Erkenntnissen aus den Expertengesprächen wurde entschieden, verschiedene Schnittstellenarten aus mehreren abgeschlossenen WMS-Kunden-Projekten zu analysieren. Dabei wurden Gemeinsamkeiten und Unterschiede identifiziert und dokumentiert, um eine Analyse durchzuführen, wie sich eine solche Schnittstelle modularisieren und generalisieren lässt, um sie für jede Schnittstellentechnologie einzusetzen und einfach anpassbar zu gestalten. Das Hauptziel ist die Gewinnung praktischer Erkenntnisse aus der realen Welt, um zusätzliche Implementierungsarten und mögliche Varianten der Umsetzung zu erforschen. Des Weiteren sollte hierbei vermieden werden, bekannte Fehler aus früheren Erfahrungen zu wiederholen.

Anschließend wurde ein Konzept zur Generalisierung dieser Gemeinsamkeiten erstellt. Dieses Konzept umfasst eine modulare, flexible und erweiterbare Struktur, die robust und zuverlässig ist. Es wurde darauf geachtet, dass die Schnittstelle leicht erweiterbar bleibt, um spezifische Kundenanforderungen zu erfüllen.

Die Praxistauglichkeit des Konzepts wurde in mehreren Schnittstellenanwendungen überprüft. Diese dienten dazu, praktische Erkenntnisse zu gewinnen und die Funktionalität zu testen. Das Feedback von Entwicklern und Anwendern wurde gesammelt und führte zu notwendigen Anpassungen und Optimierungen des Konzepts.

5.4 Konzept eines erweiterbaren Schnittstellenmodells

5.4.1 Gemeinsamkeiten in der Ein- und Ausgangsrichtung

Um Datenverluste zu vermeiden und einen umfassenden Überblick über alle ein- und ausgehenden Telegramme zu gewährleisten, soll im ersten Schritt eine Datenbanktabelle mit dem Namen *„InterfaceProtocol“* erstellt werden. Diese Tabelle sollte nur aus essenziellen Feldern bestehen, damit kein Overhead entsteht. Die Felder, die existieren sollen, sind: *„interfaceProtocolId“, „telegram“, „status“, „sentTime“, „processedTime“, „source“, „target“, „categorie“, „errorMessage“, „errorThrowsIn“* und

„*responseId*“. Das Feld „*interfaceProtocolId*“ soll als Primärschlüssel definiert werden.

Diese Tabelle dient dazu, den Überblick über die ein- und ausgehenden Telegramme zu bewahren und Daten vom Host-System sicher zu speichern, bis das System die Kapazität hat, sie zu verarbeiten. Der „*InterfaceProtocolManager*“ ist für das Schreiben der empfangenen Informationen in die „*InterfaceProtocol*“-Tabelle verantwortlich.

Die Speicherung der empfangenen Telegramme ist essenziell, um Datenverlust zu vermeiden. Ohne diese Maßnahme besteht die Gefahr, dass Informationen verloren gehen, wenn sie für eine gewisse Zeit auf der Warteliste stehen und auf ihre Verarbeitung warten. Dies ist besonders kritisch, wenn der WMS-Server neu gestartet werden muss oder im schlimmsten Fall abstürzt. In einem solchen Szenario besteht keine Möglichkeit mehr, die verlorenen Daten zu verarbeiten. Demzufolge würde das Host-System fälschlicherweise über die erfolgreiche Zustellung der Telegramme informiert, obwohl diese Daten vom WMS nicht verarbeitet wurden. Solche Probleme können zu Dateninkonsistenz zwischen den beiden Systemen führen.

Durch die Einführung dieser Zwischenspeicherung in der „*InterfaceProtocol*“-Tabelle soll sichergestellt werden, dass alle Daten zuverlässig gespeichert und verarbeitet werden können. Dies erhöht die Robustheit und Zuverlässigkeit der Schnittstelle erheblich und trägt dazu bei, Dateninkonsistenzen und Verluste zu vermeiden.

5.4.2 Besonderheiten der Eingangsrichtung

Für die Datenübertragung vom Kundensystem zum WMS wurde ein Konzept entwickelt, welches den Prozess in zwei Schritten wie abgebildet aufteilt:

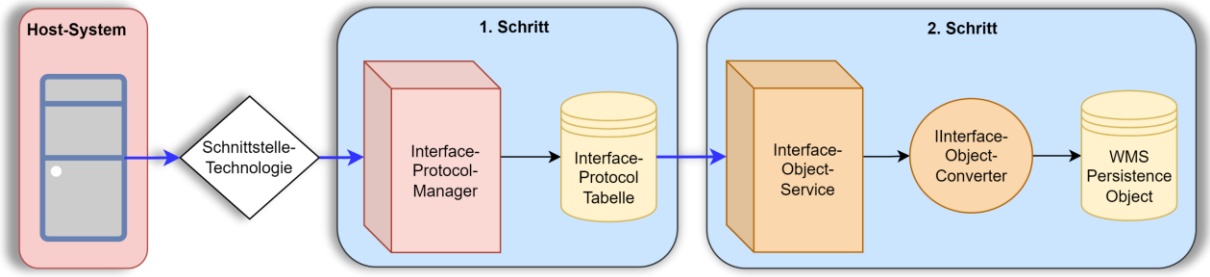


Abbildung 16 Das Konzept der generalisierten Schnittstelle für die Eingangsrichtung

Wenn das Host-System Informationen an das WMS sendet, werden diese Daten direkt in der Implementierung der Schnittstellen-Technologie zunächst validiert. Nur bei vollständiger Validität wird der Prozess fortgesetzt andernfalls wird das Host-System sofort über die Unvollständigkeit oder Fehlerhaftigkeit der Daten informiert.

Nach erfolgreicher Validierung werden die empfangenen Informationen in der Tabelle „*InterfaceProtocol“* gespeichert und das Host-System über die erfolgreiche Zustellung der Daten benachrichtigt.

Im zweiten Schritt werden die Datensätze aus der „*InterfaceProtocol*“-Tabelle mit der Quelle "*HOST*" und dem Ziel "*WMS*" in der Reihenfolge ihres Eingangs verarbeitet. Erfolgreich verarbeitete Datensätze werden mit dem Status „*BEARBEITET*“ und einem Zeitstempel versehen. Können die Daten jedoch nicht verarbeitet werden, werden sie mit dem Status „FEHLER“, dem Grund der Fehlermeldung und gegebenenfalls einem Stacktrace gekennzeichnet. Diese Vorgehensweise ermöglicht eine effiziente Entwicklung und einen effektiven Service, da Fehler im System schneller identifiziert werden können.

Dieser Prozess wird vom „*InterfaceObjectService“* im WMS übernommen. Dieser Service liest die zu verarbeitenden Datensätze mit dem Status „*ANGELEGT*“ aus der „*InterfaceProtocol“*-Tabelle aus und verarbeitet sie. Der „*InterfaceObjectService“* ist eine abgeleitete Instanz des abstrakten „*AutoService*“ im WMS, welcher in festgelegten Intervallen, die über Spring-Boot-XML konfigurierbar sind, bestimmte Aufgaben übernimmt. Ein „*AutoService“* kann bei Bedarf auch manuell durch eine Nutzeraktion in der GUI ausgelöst werden.

Bei der Verarbeitung neuer, unbearbeiteter Datensätze sucht der „*InterfaceObjectService“* anhand der Kategorie den passenden Konverter und leitet den Datensatz zur Verarbeitung an diesen weiter. Ein Konverter stellt ein Interface dar und dient als API. Klassen, die dieses Interface implementieren, müssen zwei Funktionen enthalten: „*execute“* und „*logError“*. Diese Funktionen ermöglichen die Verarbeitung der Datensätze im entsprechenden Konverter. Dabei werden die internen WMS-Persistence-Objekte entweder neu angelegt oder, falls sie bereits existieren, bearbeitet.

5.4.3 Besonderheiten der Ausgangsrichtung

Für die Übertragung der ausgehenden Informationen vom WMS an das Host-System wurde folgendes Konzept erstellt:

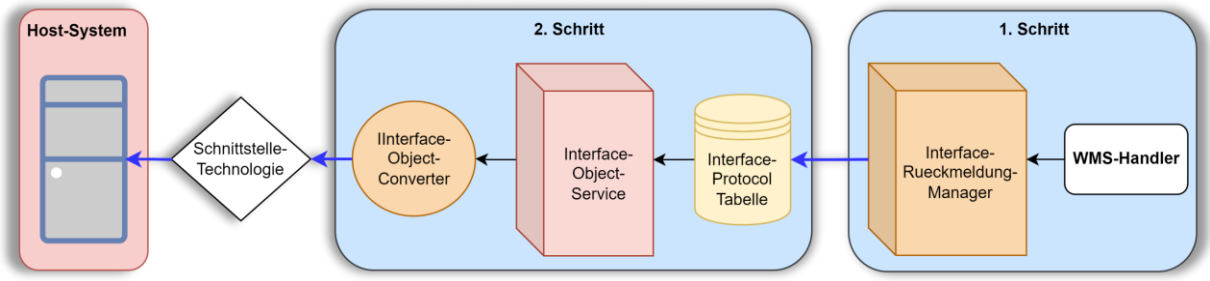


Abbildung 17 Das Konzept der generalisierten Schnittstelle für die Ausgangsrichtung

Im Gegensatz zur eingehenden Richtung, bei der das Host-System als Auslöser fungiert, ist bei der ausgehenden Richtung das WMS der Auslöser. Nach Abschluss bestimmter Prozesse wie Wareneingang und -ausgang oder Bestandsdifferenzen durch Inventuren, soll das WMS eine Meldung an das Host-System senden. Das Versenden dieser Informationen erfolgt über einen Manager namens *„InterfaceRueckmeldungManager“,* der alle notwendigen Rückmeldefunktionen des WMS enthält.

Alle Funktionen des *„InterfaceRueckmeldungManager“* sollen anpassbar und erweiterbar sein, um kundenprojektspezifische Anpassungen zu ermöglichen. Der *„InterfaceRueckmeldungManager“* nimmt ein WMS-PersistenceObject entgegen, extrahiert die essenziellen Daten, die das Host-System zur Weiterverarbeitung benötigt, und protokolliert diese Daten im JSON- oder XML-Format in der *„InterfaceProtocol“*-Tabelle. Diese Einträge werden mit der Quelle „WMS“ und dem Ziel „HOST“ versehen.

Die *„InterfaceProtocol“*-Tabelle wird vom *„InterfaceObjectService“* überwacht, wobei nur Datensätze mit der Quelle „WMS“ und dem Ziel „HOST“ berücksichtigt werden. Werden neue Datensätze gefunden, werden diese an den zuständigen Konverter weitergeleitet, der sie gegebenenfalls validiert oder formatiert und schließlich an das Host-System sendet.

Falls das Host-System zu diesem Zeitpunkt nicht erreichbar ist, soll im entsprechenden *„InterfaceProtocol“*-Datensatz der Status *„FEHLER_SENDEN“* vermerkt werden, sodass der Datensatz beim nächsten Durchlauf wieder berücksichtig wird und erneut an das Host-System gesendet wird.

5.5 Umsetzung

5.5.1 Gemeinsamkeiten in der Ein- und Ausgangsrichtung

Die Implementierung der Schnittstelle wurde generalisiert, indem ein effizientes und fehlersicheres Sitzungsmanagement für die Datenbank entwickelt wurde. Hierzu wurde eine generische Funktion namens „*databaseWork*“ im „*InterfaceObjectService*“ erstellt, die eine neue gültige Sitzung öffnet, eine Datenbankaktion unter Verwendung eines übergebenen Handlers durchführt und im Falle eines Fehlers während der gesamten Verarbeitung diesen Fehler analysiert, protokolliert und als Wert zurückgibt.

Die Funktion "*databaseWork*" vereinfacht den Datenbankzugriff und ermöglicht eine generische Nutzung sowohl für das Auslesen von „*InterfaceProtocol*“-Datensätzen als auch für andere Objekte. Diese Generalisierung trägt wesentlich zur verbesserten Wiederverwendbarkeit und Wartbarkeit des Datenbankzugriffs bei und kann von beiden Richtungen der Schnittstelle genutzt werden.

Der Quell-Code zum Aufbau dieser Funktion wird in der **Abbildung 47** im Anhang beigefügt.

Zur Befüllung der „*InterfaceProtocol*“-Tabelle wurde ein spezifischer Manager namens "*InterfaceProtocolManager*" entwickelt. Dieser Manager bietet die Funktion "*createNewProtocol*", mit der unter Angabe der Kategorie sowie der Herkunft und des Ziels eines Telegramms die entsprechenden Informationen in der Datenbank gespeichert werden. Folgende Abbildung verdeutlicht die genaue Funktionsweise dieser Operation:

```
    Erstellt einen InterfaceProtocol

    Params: category – die Kategorie des empfangenen Telegramms
            telegram – der Inhalt des empfangenen Telegramms
            interfaceSource – die Quelle des empfangenen Telegramms
            interfaceTarget – das Ziel des empfangenen Telegramms

    Returns: InterfaceProtocol

    Author: ZEN, ifd GmbH
public InterfaceProtocol createNewProtocol(@NonNull final String category, @NonNull final String telegram,
        @NonNull final String interfaceSource, @NonNull final String interfaceTarget) {
    return create(telegram, InterfaceProtocolStatus.ANGELEGT, LocalDateTime.now(), interfaceSource,
            interfaceTarget, category);
}
```

Abbildung 18 Die Funktion zum Erstellen von „*InterfaceProtocol*“ Datensätzen

Der Aufruf dieser Funktion muss vom Schnittstellenentwickler selbst an den richtigen Stellen platziert werden.

Die „*InterfaceProtocol*“-Datensätze werden dann von spezialisierten "*InterfaceObjectServices*" verarbeitet, die zuvor registrierte Datenkonverter verwenden. Diese Services überwachen die „*InterfaceProtocol“*-Tabelle und verarbeiten neue Datensätze entsprechend ihrer Kategorie durch die zugeordneten Konverter.

Die Datenkonverter sind Klassen, die das Interface "*IInterfaceObjectConverter*" implementieren müssen, welches die Funktionen "*execute*" und "logError" vorschreibt. Jeder Konverter ist auf die Verarbeitung von „*InterfaceProtocol“*-Datensätzen einer spezifischen Kategorie ausgelegt, wodurch die Zuordnung zwischen Kategorie und Konverter eindeutig sein muss.

Bei der Verarbeitung ruft der „*InterfaceObjectService“* die Funktion „*execute“* von dem Konverter auf, welcher ihm über Spring übergeben wurde. Dieser Aufruf sieht wie folgt aus:

```
    Der IInterfaceObjectConverter, der das empfangene Telegramm verarbeiten soll
protected IInterfaceObjectConverter<M> converter;

    Funktion, welche das Telegramm und die Kategorie weiter verarbeitet oder in einer Tabelle
    wegschreibt.
    Z.B. empfangene Pakete in eine Tabelle wegschreiben

    Params: mgrs – ManagerFactory
            category – die Kategorie des Telegramms
            telegram – der Inhalt des Telegramms

    Throws: Throwable – Throwables können direkt geworfen werden

    Author: ZEN, ifd GmbH
@Override
protected void write(final M mgrs, final String category, final String telegram)
        throws Throwable {

    this.converter.execute(mgrs, category, telegram);
}
```

Abbildung 19 Die Funktion „*write*“ zur Weiterleitung der Telegramme an den Konverter

Bei der Initialisierung des Servers registriert das Spring-Framework die Datenkonverter beim "*InterfaceObjectService*". Um mehrere Kategorien mit ihren Konvertern effizient zu verwalten, wurde die Klasse "*InterfaceObjectConverterCollection*" entwickelt. Diese Klasse implementiert ebenfalls das Interface "*IInterfaceObjectConverter*" und verwendet eine „*Map*“ zur Zuordnung von Kategorien zu den entsprechenden Konvertern.

Dem *„InterfaceObjectService“* wurde eine *„InterfaceObjectConverterCollection“* als Konverter zugewiesen, bei der Verarbeitung gibt er die Kategorie und das Telegramm an ihn weiter und die *„InterfaceObjectConverterCollection“* sucht in ihrer Zuordnung mit der übergebenen Kategorie nach dem passenden Konverter und ruft *„execute“* auf. Falls sie mit der übergebenen Kategorie keinen passenden Konverter findet, so wird eine Fehlermeldung wie folgt ausgelöst:

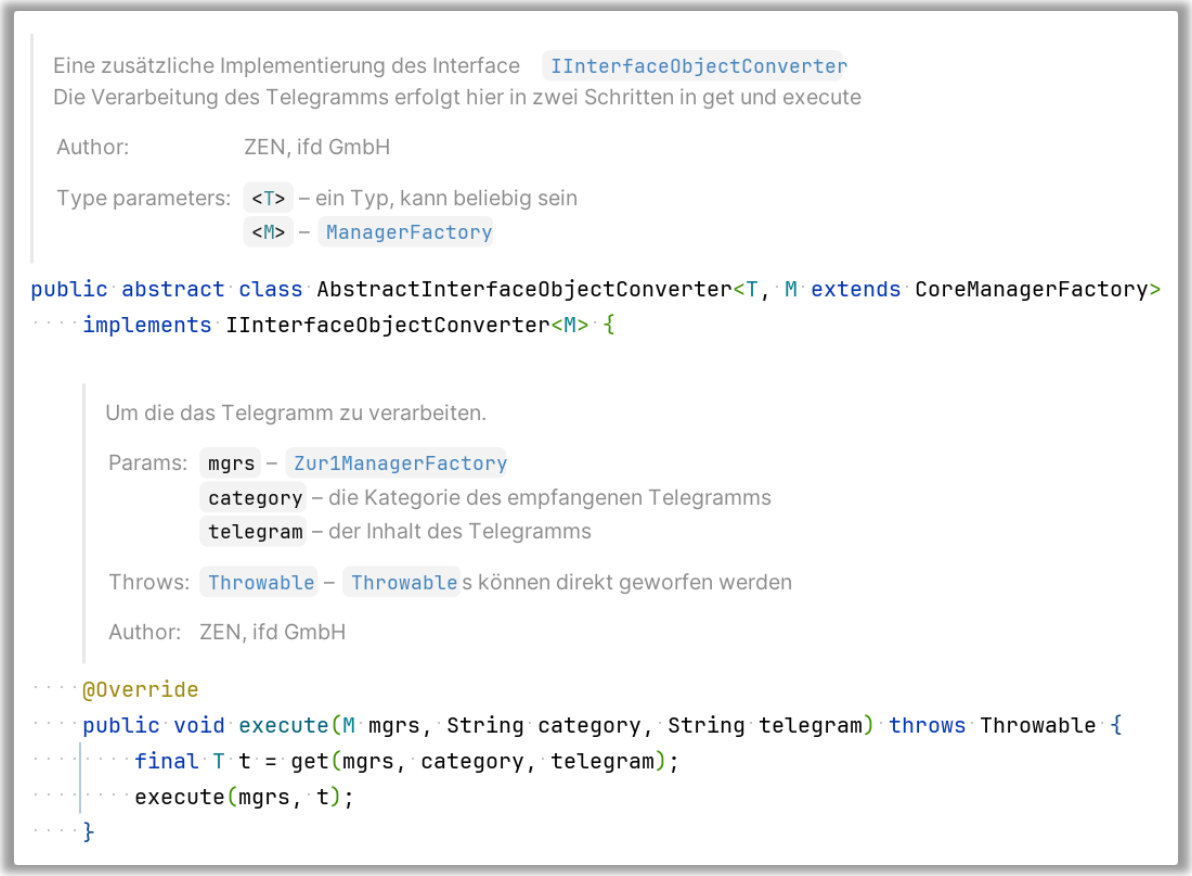
```
public class InterfaceObjectConverterCollection<M extends CoreManagerFactory>
    implements IInterfaceObjectConverter<M> {

    Zuordnung zwischen Kategorie als Key und IInterfaceObjectConverter als Value

    private final Map<String, IInterfaceObjectConverter<M>> categoryConverterMap = new HashMap<>();

    Um die Verwandlung des Telegramms in ein WMS-Object durchzuführen.

    Params: mgrs – Zur1ManagerFactory
            category – die Kategorie des empfangenen Telegramms
            telegram – der Inhalt des Telegramms

    Throws: Throwable – Throwables können direkt geworfen werden

    Author: ZEN, ifd GmbH

    @Override
    public void execute(final M mgrs, final String category, final String telegram)
            throws Throwable {

        final IInterfaceObjectConverter<M> converter = this.categoryConverterMap.get(category);

        Assert.notNull(converter, InternalError.EXCEPTION,
                String.format("Es wurde kein Konverter für diese Kategorie registriert: %s", category));

        converter.execute(mgrs, category, telegram);
    }
```

Abbildung 20 Der Aufbau der Zuordnung zwischen Kategorie und Konverter

Mithilfe dieser Technologie existiert während der Systemlaufzeit nur ein einziger *„InterfaceObjectService“*.

Um die Implementierung der Konverter weiter zu vereinfachen, wurde eine abstrakte Klasse namens "*AbstractInterfaceObjectConverter*" eingeführt. Diese abstrakte Klasse dient als Zwischenschicht für die spezifischen Konverter und implementiert die grundlegende Funktion "*execute*". Sie definiert auch abstrakte Methoden wie "*get*" und „*execute*“, die die Konverter für ihre spezifischen Aufgaben implementieren müssen.

In folgender Abbildung wird die abstrakte Klasse „*AbstractInterfaceObjectConverter*“ und ihre Implementierung der Funktion „*execute*“ gezeigt:



```
Eine zusätzliche Implementierung des Interface IInterfaceObjectConverter
Die Verarbeitung des Telegramms erfolgt hier in zwei Schritten in get und execute

Author:          ZEN, ifd GmbH

Type parameters: <T> – ein Typ, kann beliebig sein
                 <M> – ManagerFactory
public abstract class AbstractInterfaceObjectConverter<T, M extends CoreManagerFactory>
    implements IInterfaceObjectConverter<M> {

    Um die das Telegramm zu verarbeiten.

    Params:  mgrs – Zur1ManagerFactory
             category – die Kategorie des empfangenen Telegramms
             telegram – der Inhalt des Telegramms

    Throws:  Throwable – Throwables können direkt geworfen werden

    Author:  ZEN, ifd GmbH

    @Override
    public void execute(M mgrs, String category, String telegram) throws Throwable {
        final T t = get(mgrs, category, telegram);
        execute(mgrs, t);
    }
```

Abbildung 21 Die Aufteilung des Verarbeitungsprozesses in klare Schritte

Diese strukturierte Vorgehensweise teilt den Verarbeitungsprozess in klar definierte Schritte auf, indem sie die Umwandlung von Telegrammen in Objekte und deren anschließende Verarbeitung oder Rückmeldung transparent und effizient gestaltet. Außerdem hilft diese Struktur den Entwicklern dabei, ihre Aufgabe bei der Implementierung von Konvertern besser nachzuvollziehen.

Am Ende der Verarbeitung eines „*InterfaceProtocol“*-Datensatzes, wird der Status auf „*BEARBEITET*“ gesetzt und der aktuelle Verarbeitungszeitstempel aktualisiert. Folgende Funktion ist dafür zuständig:

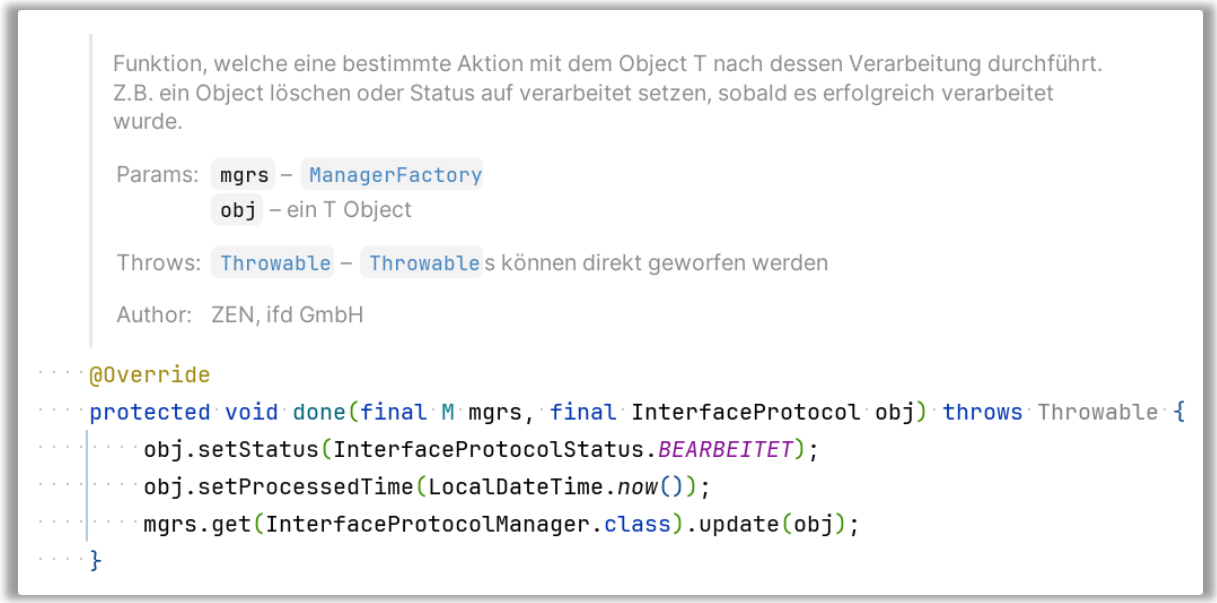


Abbildung 22 Die Funktion "*done*" zum Abschließen des Verarbeitungsprozesses

Falls eine Fehlermeldung während der gesamten Verarbeitung auftritt, wird diese Fehlermeldung analysiert, geloggt und der Datensatz als fehlerhaft vermerkt. Diese Abbildung zeigt die Funktion: **Abbildung 48**

5.5.2 Besonderheiten der Eingangsrichtung

Die empfangenen Daten werden zunächst in der „*InterfaceProtocol“*-Tabelle gespeichert, um sie vorerst sicher aufzubewahren. Anschließend wird die „*InterfaceProtocol“*-Tabelle mittels der zuvor beschriebenen „*databaseWork“*-Funktion vom „*InterfaceObjectService“* überwacht. Liegen neue Datensätze zur Verarbeitung bereit, werden diese anhand ihrer Kategorie an den zuständigen Datenkonverter weitergeleitet. Im entsprechenden Konverter werden die Informationen genauer analysiert. Falls das Objekt im WMS existiert, dann wird es aktualisiert. Falls nicht, dann wird das Objekt neu angelegt. Dabei ist die Kommunikation mit dem Kunden wichtig, welches Feld vom übertragenen Objekt eindeutig ist.

Folgender Screenshot zeigt als Beispiel eine Funktion in einem Konverter, welcher für die Erstellung oder Aktualisierung von Kundenadressen im WMS verantwortlich ist:

```
    Legt ein Land Objekt an oder aktualisiert es.

    Params:  mgrs – die ManagerFactory zur Session Verwaltung
             landKurzel – Land Kürzel

    Returns:  das neu erstellte oder das existierende und aktualisierte Land-Objekt
private Land createOrUpdate(final Zur1ManagerFactory mgrs, final String landKurzel) {
    // LandManager holen
    final LandManager<Zur1ManagerFactory, Land, VLand> landMgr = mgrs.getLandManager();

    // Land anhand des Kürzels in der DB suchen
    final Land landByKuerzel = landMgr.getByKuerzel(landKurzel);

    // Ein Mapper zum speichern oder aktualisieren erstellen
    final DBOutputMapper<Land, LandManager<Zur1ManagerFactory, Land, VLand>> landMapper =
            new DBOutputMapper<>(landMgr, landByKuerzel);

    // Save aufrufen
    return landMapper.save(() -> {

        // Vor dem Save wird das Kürzel aktualisiert oder neu gesetzt
        landMapper.updateField(LandFields.kuerzel, landKurzel);
    });
}
```

Abbildung 23 Die Funktion eines Konverters zur Erstellung oder Aktualisierung eines Land-Objekts

Die Konfiguration des „*InterfaceObjectService*“ für die Eingangsrichtung sowie die Registrierung der Konverter erfolgt über das Spring-Framework. Diese Konfiguration für die Eingangsrichtung ist wie folgt strukturiert:

```
<!-- Empfangsschnittstelle des WMS -->
<bean id="interfaceObjectService"
      class="de.ifdgmbh.core.server.manager.interfaceobject.InterfaceObjectService">
    <constructor-arg index="0" value="HIOS"/>
    <property name="sessionFactory" ref="wcsSessionFactory"/>
    <property name="managerFactoryClass"
              ref="managerFactoryClass"/>
    <property name="waitTime" value="10000"/>
    <property name="converter" ref="converterCollection"/>
    <property name="source" value="HOST"/>
</bean>

<bean id="converterCollection"
      class="de.ifdgmbh.core.server.manager.interfaceobject.InterfaceObjectConverterCollection">
    <property name="categoryConverterMap">
        <util:map id="converterMap">
            <entry key="CreateArt" value-ref="HostSendArtConverter"/>
            <entry key="CreateKndAuf" value-ref="HostSendAufConverter"/>
        </util:map>
    </property>
</bean>

<bean id="HostSendArtConverter"
      class="de.ifdgmbh.server.manager.interfaceobject.HostSendArtConverter"/>
<bean id="HostSendAufConverter"
      class="de.ifdgmbh.server.manager.interfaceobject.HostSendAufConverter"/>
```

Abbildung 24 Die Konfiguration der Empfangsschnittstelle über Spring-Framework

5.5.3 Besonderheiten der Ausgangsrichtung

Anders als bei der Eingangsrichtung, ist bei der Ausgangsrichtung das WMS der Auslöser des Prozesses. Nach Abschluss eines Prozesses wie Warenein- oder -ausgang sendet das WMS bestimmte Informationen an das Host-System. Diese relevanten Daten werden durch einen Funktionsaufruf an den „*InterfaceRückmeldungManager*“ übergeben. Die Funktion nimmt diese Daten, kapselt sie in ein Transferobjekt und serialisiert sie als Zeichenkette. Anschließend werden sie in die „*InterfaceProtocol*“-Tabelle eingetragen, zusammen mit ihrer Kategorie, Quelle und Ziel.

Im „*InterfaceRückmeldungManager*“ wurden verschiedene Funktionen implementiert, wobei jede Funktion für die Rückmeldung eines bestimmten Objekts zuständig ist. Eine Funktion zur Rückmeldung des Wareneingangs an das Host-System könnte beispielsweise wie folgt aussehen:

```
    Params: lieferant – String HostLieferant getBezeichnung()
            lsDatum – LocalDateTime WeAuf.getBelegdatum()
            lsNummer – String WeAuf.getLlsNummer()
            status – WeAufStatus WeAuf.getStatus()
            weNummer – String WeAuf.getWeNummer()
            poses – PosesType Liste mit Positionen

    Author: VME, ifd GmbH
public void rueckmeldungWareneingang(final String lieferant, final LocalDateTime lsDatum,
        final String lsNummer, final WeAufStatus status,
        final String weNummer, final List<PosesType> poses) {

        final Wareneingang wareneingang = new Wareneingang();
        wareneingang.setLieferant(lieferant);
        wareneingang.setLsDatum(IConverterHelper.localDateTime2XmlGregorianCalendar(lsDatum));
        wareneingang.setLsNummer(lsNummer);
        wareneingang.setStatus(status != null ? status.name() : null);
        wareneingang.getPoses().addAll(poses);

        writeProtocol("ConfirmWeAuf", wareneingang);
    }
```

Abbildung 25 Die Funktion zur Kapselung der zurückzumeldenden Informationen eines Wareneingangs

Die Serialisierung des Transferobjekts in XML-Form sowie das Festschreiben in die „*InterfaceProtocol*“-Tabelle erfolgt über die Funktion „*writeProtocol*“. Diese Funktion arbeitet wie folgt:

```
    Serialisiert das Objekt in XML-Form und schreibt es in die InterfaceProtocol-Tabelle

    Params: kategorie – Die Kategorie des Telegramms
            object – das zu übertragende Objekt
protected void writeProtocol(final String kategorie, final Object object) {
        try (final StringWriter stringWriter = new StringWriter()) {
            JAXB.marshal(object, stringWriter);
            stringWriter.flush();

            final InterfaceProtocolManager ipMgr = this.mgrs.get(InterfaceProtocolManager.class);

            String source = "WMS";
            String target = "HOST";
            ipMgr.createNewProtocol(kategorie, stringWriter.toString(), source, target);
        } catch (final IOException e) {
            Log.exc(this, e);
        }
    }
```

Abbildung 26 Die Funktion zur Serialisierung eines Transferobjekts in XML-Form

Ähnlich wie bei der Eingangsrichtung wird ein „*InterfaceObjectService“* die „*InterfaceProtocol“*-Tabelle überwachen und die Datensätze der Reihenfolge nach an den zuständigen registrierten Konverter weiterleiten. In den einzelnen Konvertern wird die Verbindung zum Host-System aufgebaut, das Telegramm als Transferobjekt deserialisiert und an das Host-System übermittelt. Der „*InterfaceObjectService*“ setzt den Status des „*InterfaceProtocol“*-Datensatzes auf „Bearbeitet“ und aktualisiert den aktuellen Verarbeitungszeitstempel. Bei einem Fehler wird dieser wie bereits beschrieben geloggt und der Datensatz als fehlerhaft markiert.

Falls das Host-System zum Zeitpunkt der Rückmeldung nicht erreichbar ist und keine Verbindung aufgebaut werden kann, wird ein besonderer Statuswert gesetzt. Dieser Statuswert sorgt dafür, dass nach einer bestimmten Zeit erneut versucht wird, die Verbindung aufzubauen und die betroffenen Datensätze zurückzumelden.

Die Konfiguration der Schnittstelle zum Host-System wird ebenfalls über das Spring-Framework vorgenommen. Folgende Abbildung zeigt die Spring-Konfiguration für die Rückmeldeschnittstelle vom WMS zum Host-System:

```
<!-- Rückmeldeschnittstelle des WMS -->
<bean id="interfaceRueckmeldeService"
      class="de.ifdgmbh.core.server.manager.interfaceobject.InterfaceObjectService">
    <constructor-arg index="0" value="HIOP"/>
    <property name="sessionFactory" ref="wcsSessionFactory"/>
    <property name="managerFactoryClass"
              ref="managerFactoryClass"/>
    <property name="waitTime" value="10000"/>
    <property name="converter" ref="converterRueckmeldeCollection"/>
    <property name="source" value="WMS"/>
    <property name="target" value="HOST"/>
</bean>

<bean id="converterRueckmeldeCollection"
      class="de.ifdgmbh.core.server.manager.interfaceobject.InterfaceObjectConverterCollection">
    <property name="categoryConverterMap">
        <util:map id="converterRueckmeldeMap">
            <entry key="ConfirmKndAuf" value-ref="WmsSendWarenausgangConverter"/>
            <entry key="ConfirmWeAuf" value-ref="WmsSendWareneingangConverter"/>
        </util:map>
    </property>
</bean>

<bean id="WmsSendWarenausgangConverter"
      class="de.ifdgmbh.server.manager.interfaceobject.WmsSendWarenausgangConverter"/>
<bean id="WmsSendWareneingangConverter"
      class="de.ifdgmbh.server.manager.interfaceobject.WmsSendWareneingangConverter"/>
```

Abbildung 27 Die Konfiguration der Rückmeldeschnittstelle über Spring-Framework

5.6 Tests und Validierung

5.6.1 Code-Review

Zu Beginn wurde die Generalisierung der Schnittstelle einer gründlichen Überprüfung durch Senior Developer unterzogen. Diese Schnittstellen-Experten prüften den Code auf korrekte Implementierung und Einhaltung der Spezifikationen. Das Feedback aus diesen Code-Reviews wurde genutzt, um die Schnittstelle weiter zu verbessern und sicherzustellen, dass sie den Anforderungen entspricht.

5.6.2 Einsatz in verschiedenen Schnittstellen

Die generalisierte Schnittstelle wurde erfolgreich in vier verschiedenen Unternehmenssystemen eingesetzt. Vor jeder Implementierung wurden detaillierte Tests durchgeführt, um die Funktionalität und Kompatibilität der Schnittstelle zu gewährleisten. Diese Tests umfassten sowohl funktionale als auch Integrationstests, um sicherzustellen, dass die Schnittstelle in jeder Umgebung korrekt arbeitet.

5.6.3 Tests mit SOAP-UI für Web-Service-Schnittstellen

Bei Kunden, die Web-Service-Schnittstellen verwenden, wurde SOAP-UI eingesetzt, um Mock-Services zu erstellen. Diese Mock-Services dienten dazu, Testnachrichten (Telegramme) an das WMS zu senden und vom WMS zu erhalten. Durch diese Tests konnte sichergestellt werden, dass die Schnittstelle ordnungsgemäß mit dem WMS kommuniziert und die erwarteten Daten korrekt überträgt.

5.6.4 Tests für Datenbankschnittstellen

Für Schnittstellen, die auf Datenbankebene arbeiten, wurden spezifische Tests durchgeführt. Diese Tests überprüften die Datenübertragung und -verarbeitung zwischen der Schnittstelle und den verbundenen Datenbanksystemen. Dabei wurde sichergestellt, dass alle Daten korrekt übertragen und gespeichert wurden, und dass die Schnittstelle effizient mit den Datenbanksystemen interagierte.

Durch diese umfassenden Test- und Validierungsverfahren konnte die Zuverlässigkeit und Flexibilität der generalisierten Schnittstelle bestätigt werden. Die positive Rückmeldung und die erfolgreichen Implementierungen bei verschiedenen Kunden zeigen, dass die Schnittstelle den Anforderungen gerecht wird und in unterschiedlichen Unternehmenssystemen effektiv eingesetzt werden kann.

6 Konzeption der GUI

6.1 Erklärung

Die Entwicklung einer graphischen Benutzeroberfläche ist ein wichtiger Bestandteil der Generalisierung der Schnittstelle und ein ergänzender Aspekt dieser Arbeit. In diesem Kapitel wird verdeutlicht, worauf es bei der Gestaltung einer solchen GUI ankommt und wie sie entworfen wird, um eine benutzerfreundliche und effiziente Konfiguration und Verwaltung der Schnittstelle zur Integration von Unternehmenssystem zu ermöglichen.

6.2 Anforderungen an die GUI

Bei der Definition der Anforderungen an die GUI werden sowohl funktionale als auch nicht-funktionale Anforderungen berücksichtigt:

Funktionale Anforderungen:

- **Einfache Navigation:** Die GUI muss eine intuitive Navigation ermöglichen, sodass Benutzer schnell und effizient ohne Komplikationen auf alle benötigten Funktionen zugreifen können.
- **Anlegen, Bearbeiten und Löschen von Feldern:** Die GUI muss den Benutzern die Möglichkeit bieten, Felder manuell anzulegen, zu bearbeiten und zu löschen.
- **Importieren von Klassenstrukturen mit allen Feldern:** Zur Vereinfachung und schnellen Bedienung der GUI soll der Benutzer Felder importieren können. Diese Funktion ermöglicht es, Felder initial zu laden, ohne sie einzeln anlegen zu müssen.
- **Zuordnen von Feldern:** Besonders wichtig ist die Kernfunktion des GUI-Mappers, welche Felder zwischen einem Host-System und dem WMS zusammen verknüpfen soll, sodass diese Zuordnungen von der bereits generalisierten modularen Schnittstelle genutzt werden können.
- **Einstellung der Konvertierungsart zwischen Feldern:** Die GUI muss Funktionen zur Konvertierung zwischen verschiedenen Datentypen bieten. Beispielsweise sollte es möglich sein, eine Straße und Hausnummer getrennt durch Leerzeichen in ein Feld zu mappen oder eine Zahl in eine Zeichenkette zu konvertieren.
- **Echtzeit-Validierung:** Die GUI soll eine Echtzeit-Validierung bieten. Die Speicherung einer Zuordnung soll verweigert werden, wenn der eingestellte Konverter für die betreffenden Datentypen nicht geeignet ist oder wenn die Typen der zugeordneten Felder nicht übereinstimmen, und kein passender Konverter eingestellt wurde.

- **Speicherung der Workspace:** Es sollte möglich sein, den aktuellen Workspace zu speichern, damit die Konfiguration später fortgesetzt werden kann, ohne dass vorherige Einstellungen verloren gehen. Dies ist besonders wichtig, wenn viele Felder für die Übertragung einer Kategorie oder eines Objekts berücksichtigt werden müssen.
- **Schnellsuche:** Die GUI sollte eine Schnellsuche bieten, die es Benutzern ermöglicht, schnell ein bestimmtes Feld oder dessen Zuordnungen zu finden und zu bearbeiten. Dies ist besonders hilfreich, um Änderungen schnell vorzunehmen.

Nicht-funktionale Anforderungen:

- **Benutzerfreundlichkeit:** Die GUI soll einfach zu bedienen sein, auch für Benutzer mit wenig technischer Erfahrung. Es soll eine einheitliche Farbgebung für Buttons und Eingabefelder geben, sodass der Nutzer schnell erkennt, um welche Aktion es sich handelt. Außerdem sollte die gleiche Schriftart und ähnliche Farben verwendet werden, die dem Corporate Identity Design der iFD entsprechen.
  Darüber hinaus ist es wichtig, Texte so weit wie möglich durch visuelle Darstellungen zu ersetzen, um den Benutzer bei der Bedienung der GUI zu entlasten. Icons sollten verwendet werden, die die Bedeutung eines Texts klar vermitteln. Um die Benutzerfreundlichkeit weiter zu fördern, sollen in der GUI bestimmte wichtige Aktionen durch Tastenkombinationen ausführbar sein, so dass Nutzer schnell ein Feld bzw. eine Zuordnung erstellen oder den Workspace speichern können. Dies ist besonders für Benutzer wichtig, die bevorzugen mit der Tastatur zu arbeiten.
- **Performance:** Die GUI muss auch bei großen Datenmengen schnell und reaktionsfähig bleiben. Dies bedeutet, dass die Ladezeiten kurz sein müssen und die Benutzeroberfläche flüssig auf Eingaben reagieren soll. Um dies zu gewährleisten, sollten effiziente Algorithmen verwendet werden. Zudem sollten asynchrone Datenverarbeitung und Caching-Mechanismen implementiert werden, um die Performance zu optimieren.

6.3 Evaluationsmethode

Die Entwicklung der benutzerfreundlichen GUI erfolgte unter der Anwendung mehrerer agiler und qualitativer Methoden, um tiefgehende Erkenntnisse über die Anforderungen und die Funktionsweise einer GUI zur Verwaltung einer Schnittstelle zu erlangen. Die Evaluierung der GUI wurde in mehreren Schritten durchgeführt, um sicherzustellen, dass die Anforderungen erfüllt werden:

- **Prototyping:** Es wurden mehrere Prototypen entwickelt, um frühzeitig Designprobleme zu identifizieren und zu beheben. Diese Prototypen halfen, die grundlegende Benutzerführung und Funktionalität der GUI zu testen und anzupassen.
- **Usability-Tests:** Um die Benutzerfreundlichkeit und Funktionalität der GUI zu bewerten, wurden Usability-Tests mit potenziellen Endbenutzern durchgeführt. Diese Tests ermöglichten es, das tatsächliche Benutzerverhalten zu beobachten und wertvolle Einblicke in die Nutzererfahrung zu gewinnen. Die Teilnehmer wurden gebeten, typische Aufgaben innerhalb der GUI zu erledigen, während ihre Interaktionen und Rückmeldungen aufgezeichnet und analysiert wurden.
- **Feedback-Runden:** Regelmäßige Feedback-Runden mit Stakeholdern und Endbenutzern wurden organisiert, um kontinuierliche Verbesserungen basierend auf tatsächlichen Nutzerbedürfnissen vorzunehmen.
- **Leistungsanalyse:** Die Leistung der GUI wurde durch das gleichzeitige Importieren von vielen Objekten, Feldern und Zuordnungen getestet. Ziel war es, sicherzustellen, dass die GUI auch unter hoher Last performant und reaktionsfähig bleibt.
- **Iterative Entwicklung:** Basierend auf den Ergebnissen der Tests und dem kontinuierlichen Feedback wurde die GUI in mehreren Iterationen weiterentwickelt. Jede Iteration zielte darauf ab, die Benutzerfreundlichkeit, Funktionalität und Leistung der GUI zu verbessern und die identifizierten Probleme zu lösen.

Durch diese systematische und iterative Evaluationsmethode konnte sichergestellt werden, dass die entwickelte GUI den Anforderungen entspricht und eine effiziente und benutzerfreundliche Verwaltung der Schnittstelle ermöglicht.

6.4 Entwurf des Layouts und der Benutzerführung

Der Entwurf des Layouts und der Benutzerführung zielt darauf ab, eine intuitive und effiziente Benutzererfahrung zu schaffen. Zu Beginn wurde ein einfacher initialer Entwurf erstellt, um einen Ausgangspunkt für weitere Diskussionen und Verfeinerungen zu haben.

Der erste Entwurf sah wie folgt aus:

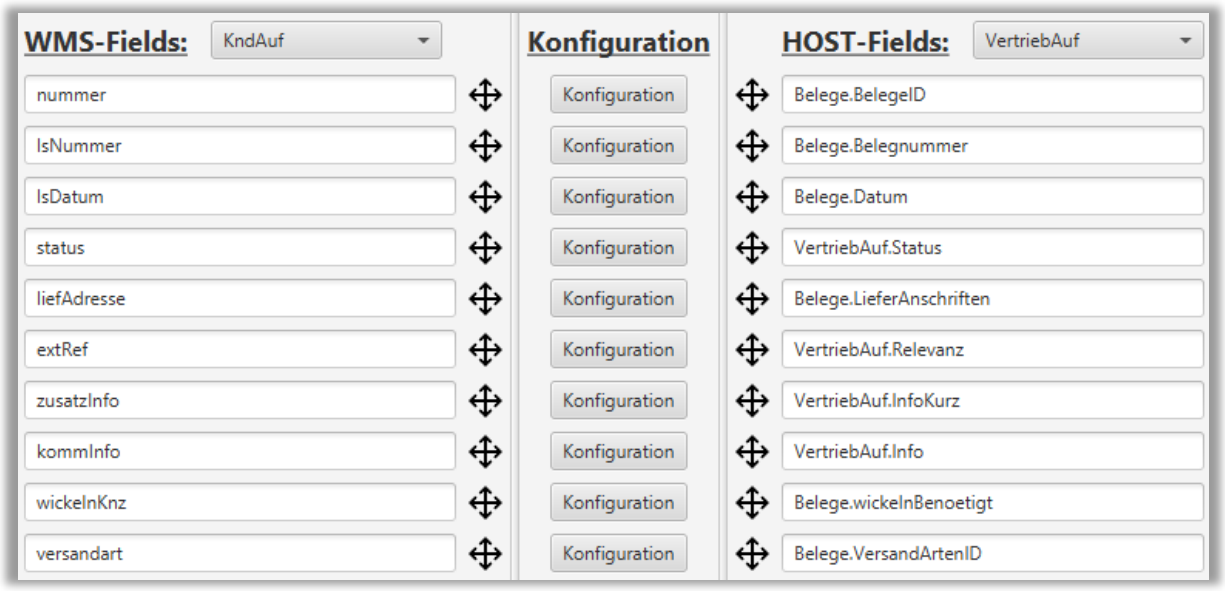


Abbildung 28 Der erste Entwurf des GUI-Mappers

Auf der linken Seite sind die Felder eines bestimmten WMS-Objekts dargestellt, welches sich mittels einer Auswahlbox ändern lässt. Auf der rechten Seite sind die Felder eines Vertrieb-Auftrags im Host-System abgebildet. Damit Felder zusammen zugeordnet werden können, müssen sie in einer gemeinsamen Zeile existieren. Im mittleren Bereich kann die Zuordnung konfiguriert werden, beispielsweise die Konvertierung einer Zeichenkette in eine dezimale Nummer.

Nach dem ersten Usability-Test und der Feedbackrunde wurde festgestellt, dass diese Struktur unnötig viel Platz auf dem Monitor beansprucht und keine hierarchische Struktur (Eltern-Kind-Struktur) unterstützt. Dadurch wurde die Funktionalität eingeschränkt. Es wurde infolgedessen entschieden, einen neuen Entwurf zu entwickeln, der die Hierarchie der Objekte darstellen kann. Basierend auf den Ergebnissen der Tests und dem kontinuierlichen Feedback wurde ein neuer Entwurf erstellt.

Folgende Abbildung veranschaulicht den zweiten Entwurf:

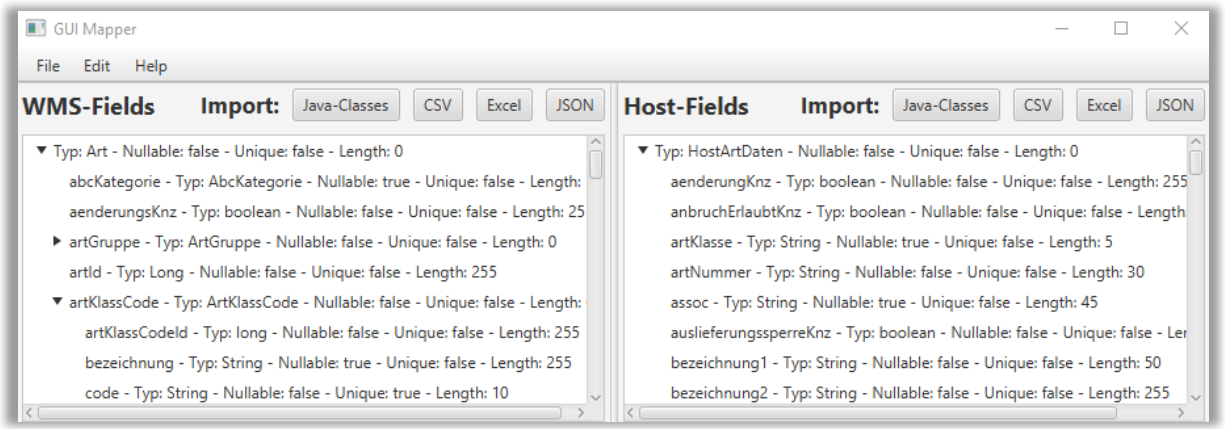


Abbildung 29 Der zweite Entwurf des GUI-Mappers

Der neue Entwurf integriert eine Baum-Struktur zur Darstellung von Hierarchien und ermöglicht eine effizientere Nutzung des Platzes auf dem Bildschirm. Durch die Überarbeitung des Layouts konnten die Benutzerfreundlichkeit und die Funktionalität der GUI erheblich verbessert werden.

Außerdem wurde im neuen Entwurf eine Import-Funktion integriert. Diese Funktion ermöglicht es, Felder initial zu laden, ohne sie einzeln anlegen zu müssen. Eine nachträgliche Anpassung des gesamten Schemas ist weiterhin möglich.

Nach einem wiederholten Test fiel dem UI-Tester auf, dass die Vielzahl der Buttons zur Import-Funktion die Klarheit und Sauberkeit der Benutzeroberfläche beeinträchtigt. Daraufhin wurde der Entwurf erneut angepasst. Es wurde ein Wizard für die Import-Funktionen implementiert, der alle Funktionen in einem „*MenuButton*“ auflistet, anstatt sie alle nebeneinander anzuzeigen. Dies stellt sicher, dass der Nutzer nicht durch eine Vielzahl von Buttons verwirrt wird und die GUI klarer und übersichtlicher bleibt.

Folgende Abbildung verdeutlicht die Verbesserung der Gestaltung der Import-Funktionen:

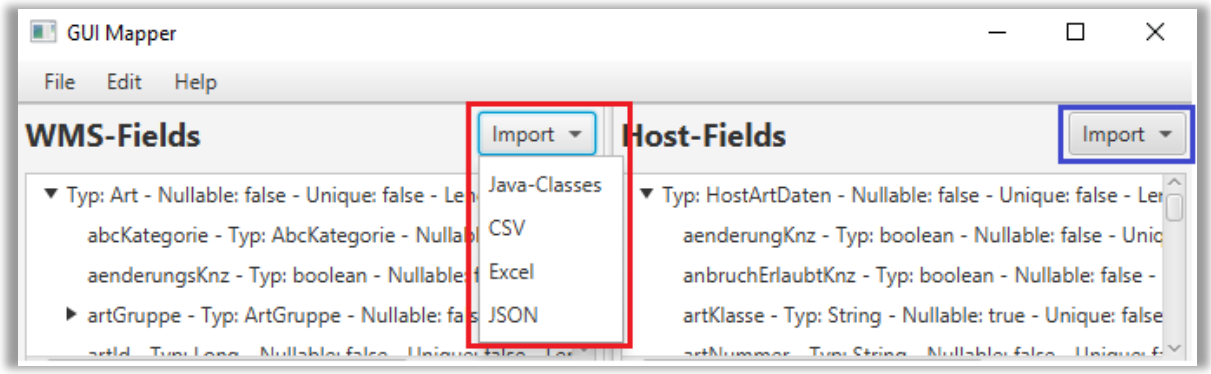


Abbildung 30 Die Verbesserung der Import-Funktion im zweiten Entwurf des GUI-Mappers

Im blauen Bereich ist der Zustand dargestellt, wenn der Nutzer nicht auf den Menübutton klickt, und im roten Bereich ist die Ansicht nach einem Mausklick auf das Menü zu sehen.

Nachdem die Import-Funktionen erfolgreich implementiert wurden, wurden die Funktionen zum Anlegen, Bearbeiten und Löschen eines Feldes entwickelt. Zum Anlegen und Löschen wurde ein „*ContextMenu*“ für die Felder in den Baum-Ansichten für die WMS- und Host-Felder entwickelt, die mittels eines rechten Mausklicks auf ein bestimmtes Feld eingeblendet wird:

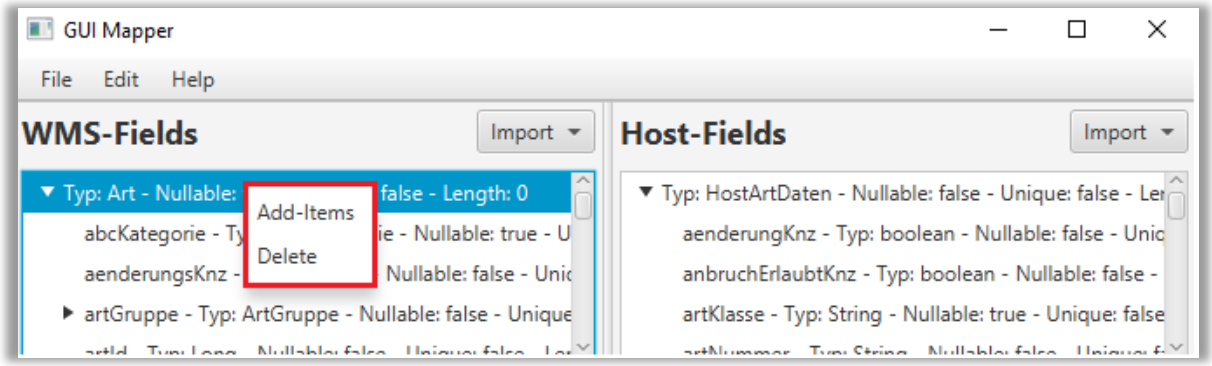


Abbildung 31 Die Funktionen zum Anlegen und Löschen eines Feldes über eine „*ContextMenu*“

Beim Anlegen öffnet sich ein Popup-Fenster, in dem die Eingabe der erforderlichen Informationen eines Feldes wie Name, Typ und Länge erfolgt. Das Fenster lässt sich komplett mit der Tastatur bedienen, wobei das Wechseln zwischen den Eingabefeldern wie üblich über die Tab-Taste und das Speichern über die Enter-Taste erfolgt. Das Fenster sieht wie folgt aus:

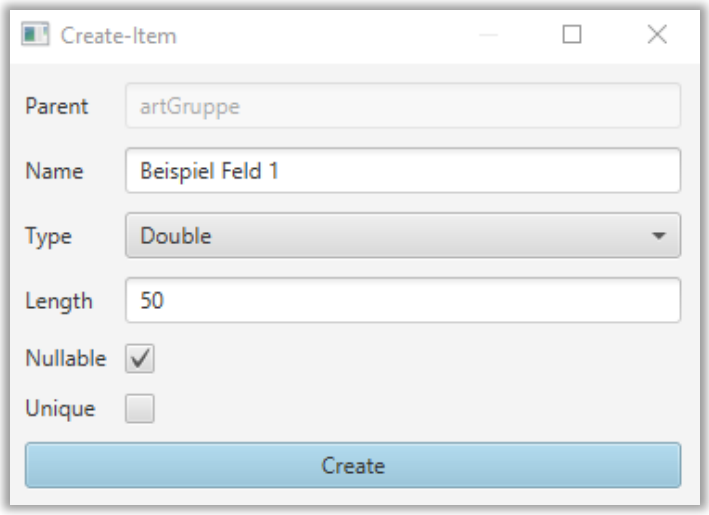


Abbildung 32 Das Popup-Fenster zum Anlegen eines Feldes

Zum Bearbeiten eines Feldes ist nur ein Doppelklick auf ein Feld nötig. Dadurch öffnet sich, wie beim Anlegen, das gleiche Fenster, jedoch im Update-Modus. Der Name des Feldes wird dabei direkt ausgewählt, damit der Nutzer den Namen nicht erst markieren muss, falls er ihn ändern möchte. Folgende Abbildung zeigt das Popup-Fenster zum Bearbeiten eines Feldes:

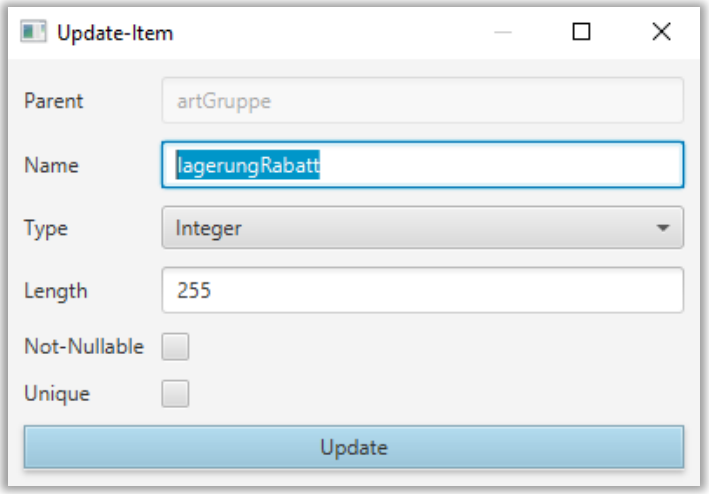


Abbildung 33 Das Popup-Fenster zum Bearbeiten eines Feldes

Zum Löschen eines Feldes wird das entsprechende „*ContextMenu*“-Item geklickt, woraufhin ein Fenster zur Bestätigung der Löschaktion erscheint, wie folgt:

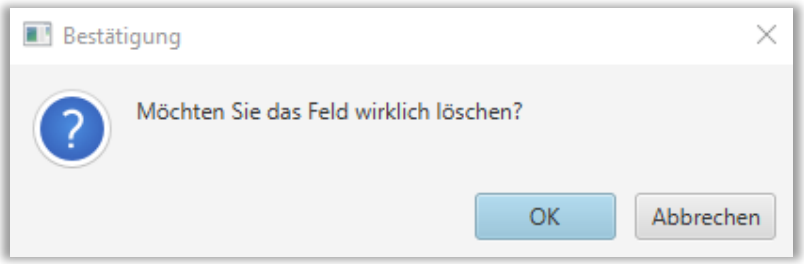


Abbildung 34 Das Popup-Fenster zur Löschbestätigung eines Feldes

Mit der Aufforderung zur Bestätigung wird ein versehentliches Löschen verhindert.

Als nächste wurde der Entwurf um eine Zuordnungstabelle erweitert. Diese Tabelle soll die bereits zugeordneten Felder beinhalten. Jede Zuordnung besitzt eine eindeutige ID. Die Zuordnungstabelle sieht wie folgt aus:

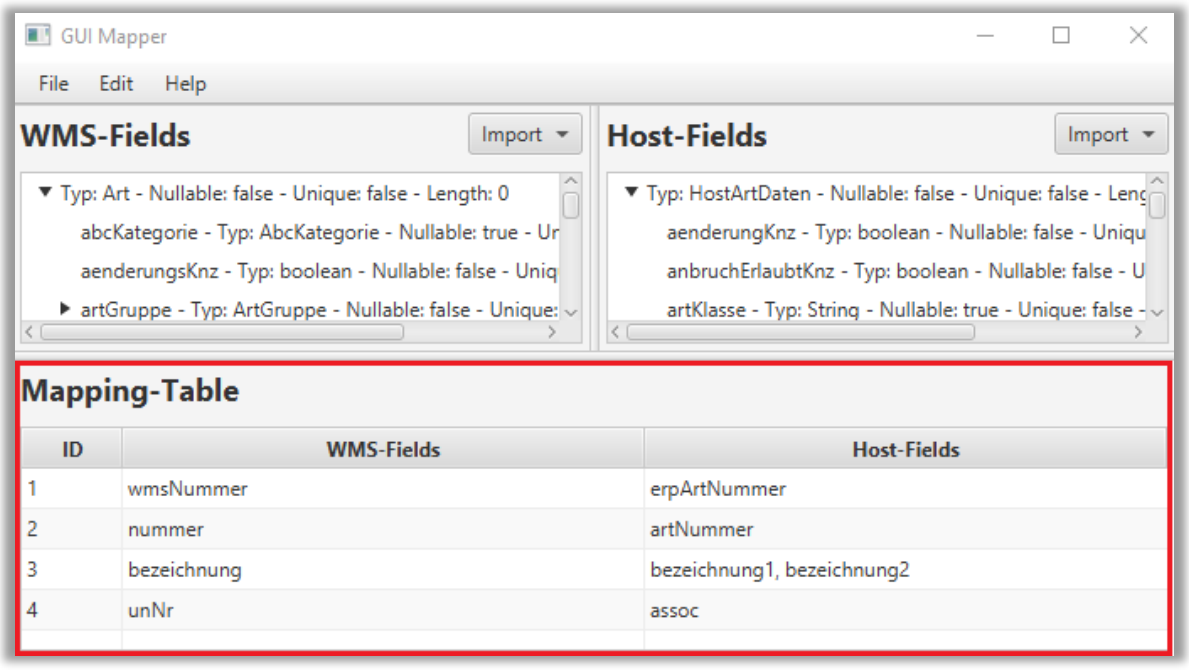


Abbildung 35 Die Darstellung der Mapping-Tabelle

Um eine Zuordnung zwischen mehreren Feldern anzulegen, müssen die zuzuordnenden Felder auf beiden Seiten angeklickt werden. Eine Zuordnung lässt sich über die Tastenkombination Strg+D anlegen, damit der Nutzer schnell die Felder selektiert und die Tastenkombination drückt, ohne zusätzlich in einen anderen Bereich mit der Maus navigieren zu müssen.

Anschließend wurde die GUI von den Testern erneut genauer untersucht und es wurden Besprechungen durchgeführt, um weiteres Feedback zu sammeln. Es wur-

den zwei Aspekte angemerkt: Einerseits, dass die Felder zu viele Informationen beinhalten, was den Testern viel Leseaufwand verursacht. Andererseits wurde gefragt, ob eine Filterung in der Mapping-Tabelle möglich wäre, damit bei vielen Zuordnungen eine bestimmte Zuordnung schneller gefunden werden kann.

Daraufhin wurde der Entwurf überarbeitet und ein großer Teil des Texts durch visuelle Darstellungen ersetzt. Es werden nun nur die essenziellen Informationen für die Zuordnung in den beiden Baum-Ansichten rechts und links eingeblendet. Der Name des Feldes wurde fett und der Typ daneben kursiv markiert. Damit wird es für den Anwender übersichtlicher.

Nach der Einführung der visuellen Darstellungen sieht der GUI-Mapper nun wie folgt aus:

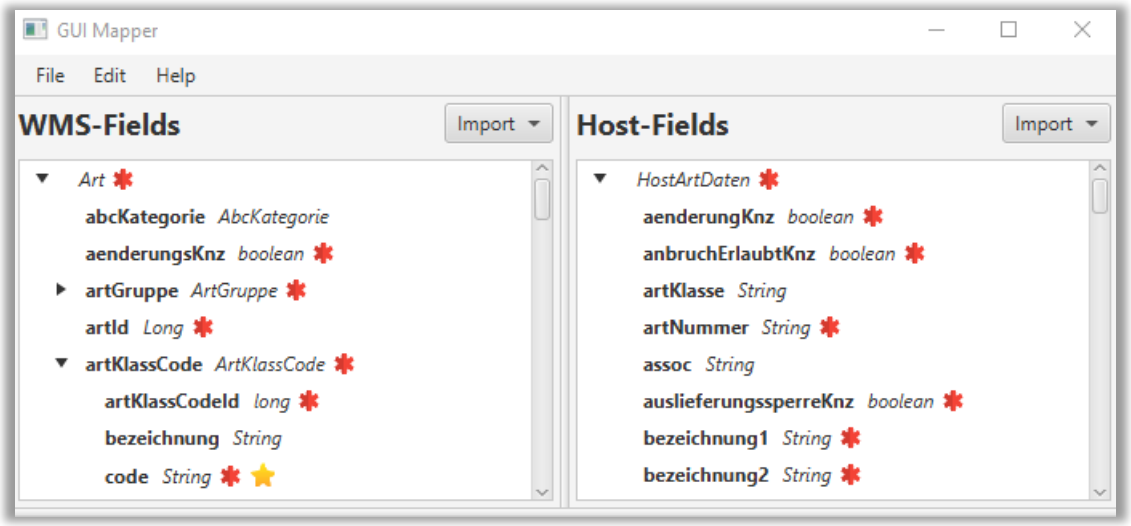


Abbildung 36 Die Erweiterung des zweiten Entwurfs um visuelle Darstellungen

Die roten Sterne identifizieren, ob ein Feld verpflichtend ist (nicht leer bleiben darf), und die gelben Sterne kennzeichnen die Eindeutigkeit eines Feldes. Die Eindeutigkeit bedeutet, dass jeder Eintrag in diesem Feld einen einzigartigen Wert haben muss, ohne Wiederholungen oder Duplikate. Der Name des Feldes wird fett angezeigt und der Datentyp kursiv daneben, damit der Nutzer die Informationen schnellstmöglich erkennt.

Zusätzlich wurden Filter-Funktionen in die Spalten der Zuordnungstabelle integriert, damit der Nutzer seine gewünschten Zeilen schnell filtern kann.

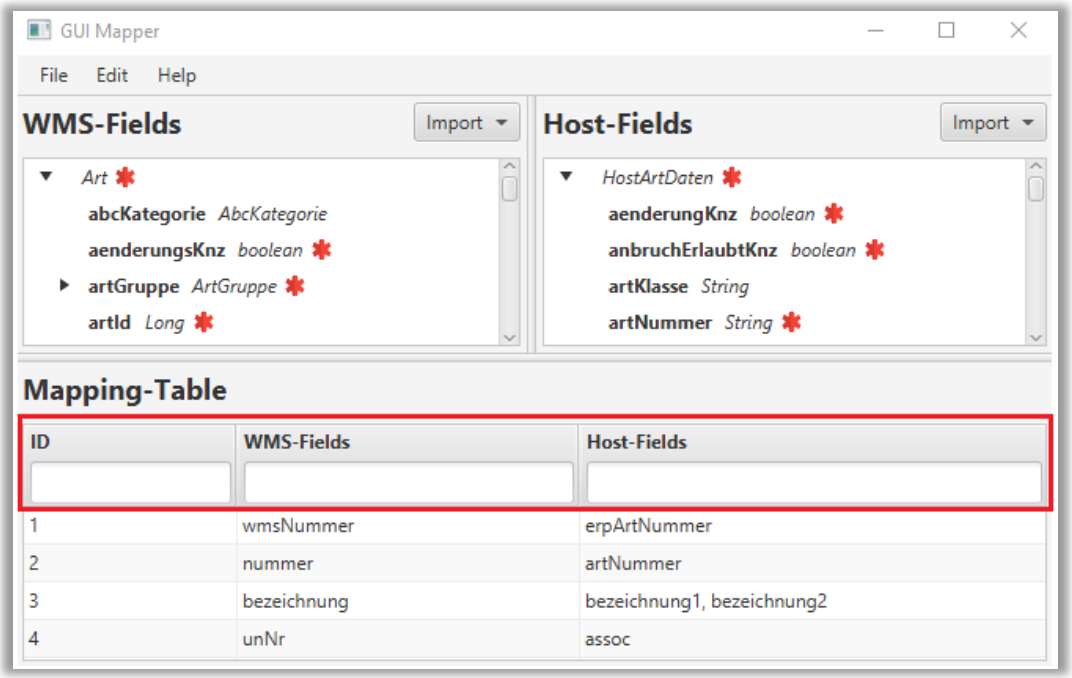


Abbildung 37 Die Filterung der Zuordnung-Tabelle

Wenn eine Eingabe in die Filterfelder getätigt wird, sucht die GUI nach dem passenden Eintrag in der Tabelle und zeigt diesen an.

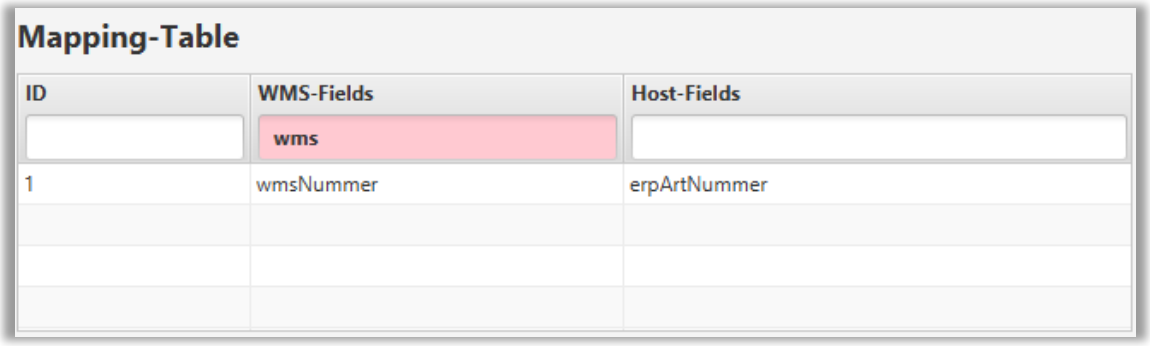


Abbildung 38 Ein Beispiel zur Filterung der Zuordnung-Tabelle

Um die Benutzerfreundlichkeit weiter zu erhöhen und die Übersichtlichkeit zu verbessern, wurde eine Funktion entwickelt, die es ermöglicht, bei einem Klick auf einen Eintrag in der Mapping-Tabelle die zugeordneten Felder auf beiden Seiten in Hellblau zu markieren:

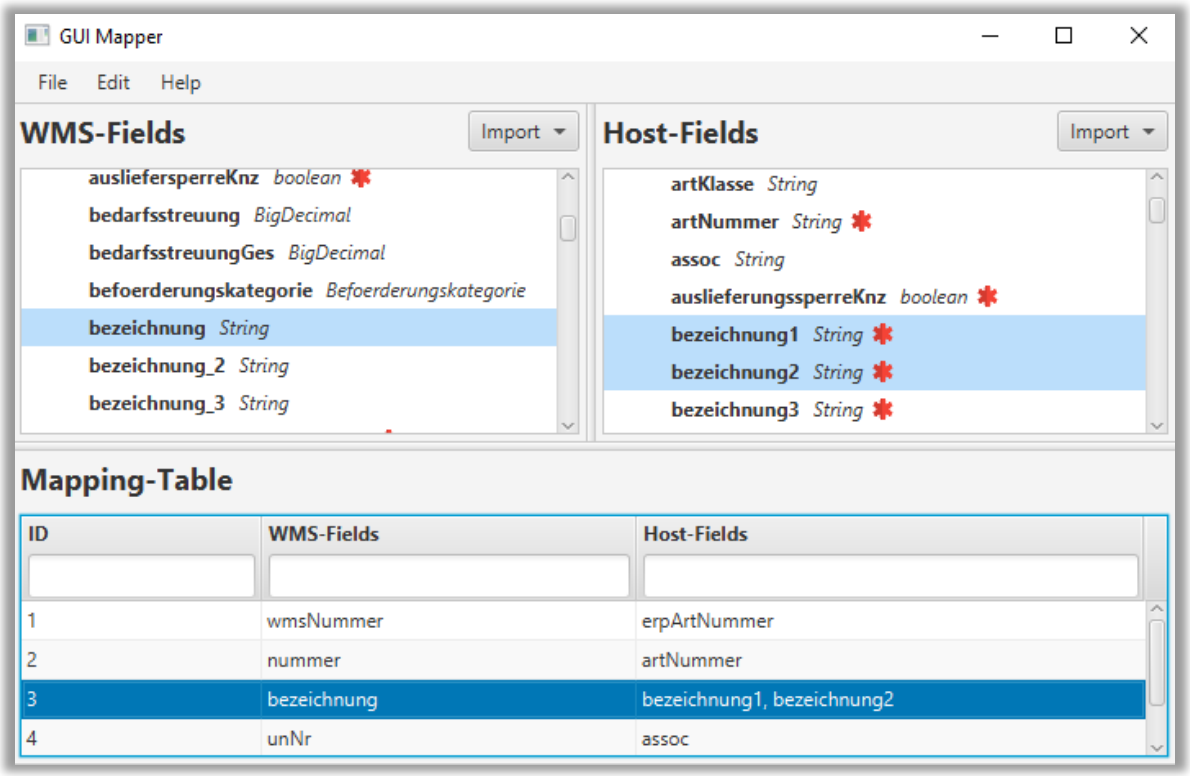


Abbildung 39 Die Auswahl einer Zuordnung und Markierung der zugehörigen Felder

Nach einer weiteren Feedback-Runde wurde angemerkt, dass der Nutzer zu viel Zeit braucht, um ein bestimmtes Feld zu finden. Daraufhin wurde eine Schnellsuche in der GUI implementiert, die sich üblicherweise über die Tastenkombination Strg+F öffnen lässt.

Die Schnellsuche ist ein Fenster, das zwei Felder beinhaltet: Mode und Text. Über diese Felder kann der Nutzer schnell das gewünschte Feld finden. Es sieht wie folgt aus:

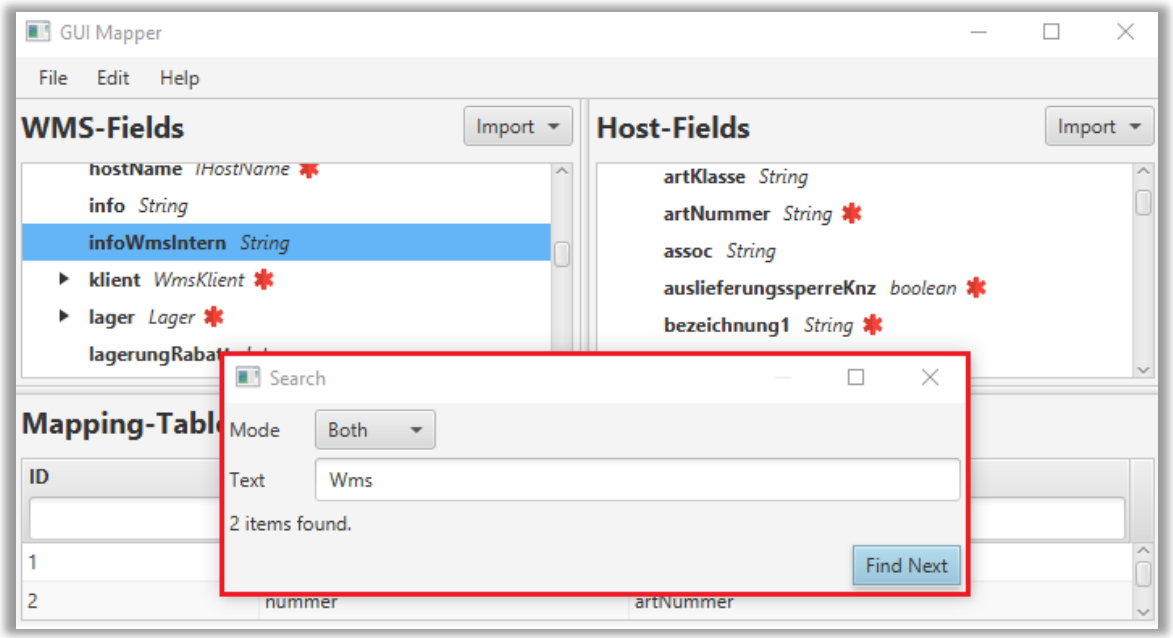


Abbildung 40 Die Schnellsuche in der GUI

Über den Mode lässt sich einstellen, in welchem Bereich der Nutzer genau suchen möchte. Zur Auswahl stehen drei Optionen: WMS, Host und Both (Beides). Zusätzlich werden die insgesamt relevanten Felder gezählt und dem Nutzer mitgeteilt. Bei einem Klick auf den Button „Find Next“ sucht der GUI-Mapper automatisch nach dem nächsten Feld, wählt es aus und scrollt zu diesem Feld, damit der Nutzer es sofort sieht.

Falls der Nutzer weitere Details zu einer Zuordnung benötigt oder Einstellungen an der Zuordnung, wie die Auswahl des Konverters, vornehmen möchte, kann er durch einen Doppelklick auf die betroffene Zuordnung ein Fenster öffnen, in dem er mehr Details einsehen und Einstellungen vornehmen kann.

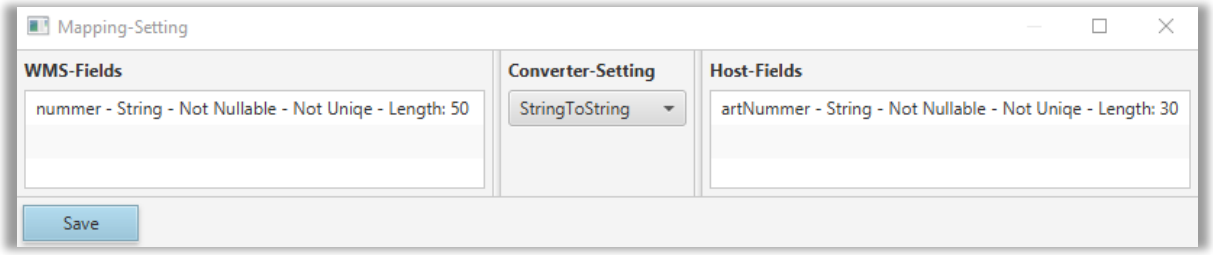


Abbildung 41 Das Fenster zur Konverter-Einstellung einer Zuordnung

Die Konverter müssen vom Entwickler einmalig entwickelt und im System registriert werden, damit die GUI diese Konverter erkennt. Zudem sollen nur die Konverter angezeigt werden, die zu den zugeordneten Feldern passen. Dies verhindert, dass der Nutzer versehentlich einen falschen oder inkompatiblen Konverter auswählt und so-

mit das System beeinträchtigt. Ein Beispiel der möglichen Konverter zeigt die folgende Abbildung:

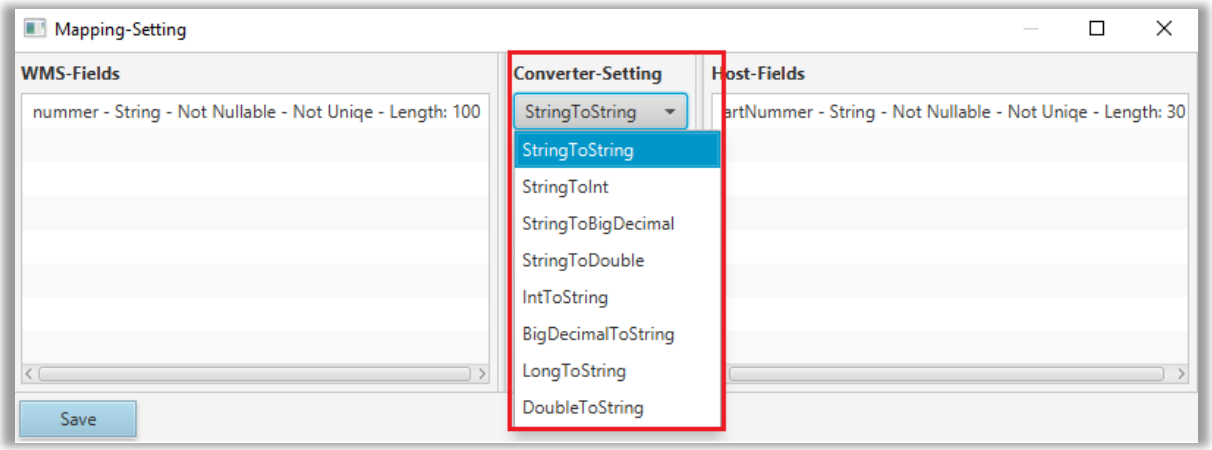


Abbildung 42 Die Konverter-Einstellungen im GUI-Mapper

Der gesamte Workspace mit den gesamten Einstellungen wird über die übliche Tastenkombination Strg+S in JSON-Form serialisiert und in einem beliebigen Ordner, den der Nutzer vorher auswählt, als Datei gespeichert.

Die gespeicherten Schnittstellen-Konfigurationen sollten künftig in der generalisierten Schnittstelle berücksichtig werden. Im automatisiert ausgeführten Service (AutoService) „*InterfaceObjectService*“ sollen die Konfigurationen mit einer entsprechenden Kategorie in einer „*Map*“ registriert werden. Der „*InterfaceObjectService*“ in der generalisierten Schnittstelle soll den JSON-Text aus der Konfigurationsdatei in ein Java-Objekt deserialisieren, um die zugeordneten Felder zu erkennen. Anschließend soll der „*InterfaceObjectService*“ eine Instanz des eingestellten Konverters mittels Java-Reflection erstellen und den Wert aus dem Feld des Quellsystems mithilfe des Konverters in das zugehörige Feld des Zielsystems transformieren.

Durch diese iterativen Verbesserungen wurde eine benutzerfreundliche und effiziente GUI konzeptioniert, die den Anforderungen der Nutzer gerecht wird und eine einfache Verwaltung der Schnittstelle zur Integration von Unternehmenssystemen ermöglicht.

6.5 Umsetzung

Die gesamte Konzeption und Entwicklung der GUI erfolgte unter Verwendung von Java und JavaFX. Diese Technologien wurden gewählt, da das WMS der iFD ebenfalls mit Java und JavaFX entwickelt wurde und andere Entwickler das neue System verstehen und es warten können.

Für die Darstellung der Felder auf beiden Seiten wurde das JavaFX-Element „*TreeView*“ eingesetzt. Ein „*TreeView*“ benötigt ein „*Root-TreeItem*“, aus dem die gesamte Baumstruktur der Felder aufgebaut wird. Sowohl die „*TreeView*“ als auch die TreeItems verlangen einen generischen Datentyp. Im GUI-Mapper wurde hierfür eine Klasse namens „*TreeItemData*“ entwickelt, die alle notwendigen Eigenschaften eines TreeItems umfasst und speichert. Die Struktur der Klasse „*TreeItemData*“ mit ihren Eigenschaften ist in der Abbildung zu sehen: **Abbildung 49**

Die Zuordnungen zwischen mehreren Feldern beider Systeme werden in der Klasse „*MappedItems*“ gespeichert. Diese Klasse enthält eine eindeutige Mapping-ID, die zugeordneten Felder sowohl auf der WMS- als auch auf der Host-Seite sowie den eingestellten Konverter. Die Klasse wird in dieser Grafik gezeigt:

```
Die Main-Factory, welche doe gesamten Daten eines Workspace beinhaltet.

Author:  ZEN, ifd GmbH
public class MappedItems implements Serializable {

    private static final long serialVersionUID = 1L;

    private int                 mappingId;
    private List<TreeItemData> wmsFields;
    private List<TreeItemData> hostFields;
    private String              converter;

    public MappedItems(@JsonProperty("mappingId") final int mappingId,
            @JsonProperty("wmsFields") final List<TreeItemData> wmsFields,
            @JsonProperty("hostFields") final List<TreeItemData> hostFields,
            @JsonProperty("converter") final String converter) {
        super();
        this.mappingId = mappingId;
        this.wmsFields = wmsFields;
        this.hostFields = hostFields;
        this.converter = converter;
    }
```

Abbildung 43 Die Klasse „*MappedItems*“ zur Speicherung von Zuordnungen

Die gesamten Daten, wie WMS-Felder, Host-Felder und Mapping-Items, werden in einer Klasse namens „*MainFactory*“ im Arbeitsspeicher verwaltet. Diese Klasse umfasst Listen der vorhandenen WMS- und Host-Felder sowie eine Liste der Zuordnungen dieser Felder.

Die Struktur der „*MainFactory*“ ist wie folgt dargestellt:

```
Die Klasse, welche die gesamten Daten eines Workspace beinhaltet.

Author:  ZEN, ifd GmbH
public class MainFactory implements Serializable {

    private static final long serialVersionUID = 1L;

    private List<TreeItemData> wmsItems;
    private List<TreeItemData> hostItems;
    private List<MappedItems>  mappedItems;

    public MainFactory(@JsonProperty("wmsParents") final List<TreeItemData> wmsParents,
            @JsonProperty("hostParents") final List<TreeItemData> hostParents,
            @JsonProperty("mappedItems") final List<MappedItems> mappedItems) {
        super();
        this.wmsItems = wmsParents;
        this.hostItems = hostParents;
        this.mappedItems = mappedItems;
    }
```

Abbildung 44 Die Struktur der gesamten Daten des GUI-Mappers „*MainFactory*“

Alle Klassen müssen serialisierbar sein, um sie später in JSON-Form speichern zu können. Aus diesem Grund implementiert jede Klasse das Interface „*Serializable*“.

Die Import-Funktion von Java-Klassen wurde durch eine Methode namens „*importJavaClasses*“ realisiert. Diese Methode nimmt eine Java-Klasse und ggf. ein Field-Objekt entgegen, untersucht alle Felder der übergebenen Klasse mittels Java-Reflection und fügt die „*TreeItems*“ mit ihren Parametern aus „*TreeItemData*“ in die entsprechende „*TreeView*“ ein. Wenn ein Feld erkannt wird, das selbst ein Persistenz-Objekt ist, erfolgt ein rekursiver Aufruf dieser Methode, um auch die Felder dieses Objekts zu untersuchen und als untergeordnete Elemente (Children) an das Haupt-Item zu hängen. Dabei wird sichergestellt, dass bereits besuchte Klassen ausgeschlossen werden, um Endlosschleifen zu vermeiden, die zum Absturz des Programms führen könnten.

Sobald der Nutzer den aktuellen Datenstand speichern möchte, öffnet sich ein Popup-Fenster, in dem er den Speicherort auswählen kann. Abschließend wird die gesamte „*MainFactory*“ mit allen Feldern und Zuordnungen serialisiert und als JSON-Datei im ausgewählten Ordner abgelegt:

```java
@FXML
public void miSaveOnAction() {
    // Überprüfen, ob die Listen von WmsItems und HostItems leer sind.
    // Wenn ja, wird eine NullPointerException geworfen.
    final MainFactory factory = this.mainFactory.get();
    if (CollectionHelper.isEmpty(factory.getWmsItems()) ||
            CollectionHelper.isEmpty(factory.getHostItems())) {
        throw new NullPointerException("Workspace empty!");
    }

    // Wenn keine Projektdatei ausgewählt wurde, öffnet sich ein Dateiauswahl-Dialog.
    if (this.selectedFile == null) {
        this.selectedFile = GuiMapperFileSelector.getSelectedFolder(".", "json");
    }
    // Wenn immer noch keine Projektdatei ausgewählt wurde, wird die Methode abgebrochen.
    if (this.selectedFile == null) {
        return;
    }

    try {
        // Erstellen eines ObjectMapper zum Serialisieren von Objekten in JSON.
        final ObjectMapper objectMapper = new ObjectMapper();
        objectMapper.findAndRegisterModules(); // Registrieren aller verfügbaren Module.

        // Serialisieren des mainFactory-Objekts in einen JSON-String mit Pretty-Printing.
        final String body = objectMapper.writerWithDefaultPrettyPrinter().writeValueAsString(factory);

        // Serialisieren des mainFactory-Objekts und direktes Schreiben in die ausgewählte Datei.
        objectMapper.writerWithDefaultPrettyPrinter().writeValue(this.selectedFile, factory);
    } catch (final Exception e) {
        // Fehler beim Speichern des Arbeitsbereichs wird geloggt.
        this.log.error("Error while saving workspace!", e);
    }
}
```

Abbildung 45 Die Funktion "*miSaveOnAction*" zum Speichern des Datenstandes

Beim Laden der gespeicherten JSON-Datei wird die Datei in die „*MainFactory*“ deserialisiert. Anschließend werden die gespeicherten Felder und Zuordnungen in die GUI geladen, sodass der Nutzer die Bearbeitung fortsetzen kann.

Diese und weitere Funktionen und Klassen wurden entwickelt, um sie für die zukünftige weitere Implementierung bereitzustellen und die Entwürfe dieser GUI auf Benutzerfreundlichkeit und andere Aspekte durch Endbenutzer testen zu können.

6.6 Tests und Validierung

6.6.1 Schwerpunkte

Die Tests und Validierungsmaßnahmen konzentrierten sich darauf, sicherzustellen, dass die entworfenen Konzepte und entwickelten Prototypen den Anforderungen entsprechen und eine solide Grundlage für die weitere Entwicklung bieten.

6.6.2 Funktionale Tests

Funktionale Tests wurden an den entwickelten Prototypen durchgeführt, um sicherzustellen, dass die grundlegenden Funktionen der GUI korrekt implementiert und funktionsfähig sind. Dies umfasste:

- **Anlegen, Bearbeiten und Löschen von Feldern:** Es wurde sichergestellt, dass die grundlegenden Funktionen zum Anlegen, Bearbeiten und Löschen von Feldern im Prototyp funktionieren.
- **Importieren von Klassenstrukturen:** Die Import-Funktion im Prototyp wurde intensiv getestet, um zu verifizieren, dass Felder korrekt importiert und angepasst werden können.
- **Zuordnen von Feldern:** Es wurde überprüft, ob die Prototyp-Funktion zur Zuordnung von Feldern zwischen dem Host-System und dem WMS korrekt funktioniert.
- **Echtzeit-Validierung:** Es wurde sichergestellt, dass die Echtzeit-Validierungsfunktionen in den Prototypen korrekt implementiert sind.

6.6.3 Usability-Tests

Usability-Tests wurden mit den Prototypen durchgeführt, um die Benutzerfreundlichkeit und Effizienz des GUI-Designs zu bewerten. Dabei wurden potenzielle Endbenutzer eingeladen, typische Aufgaben innerhalb der Prototyp-GUI zu erledigen und Feedback zu geben. Die wichtigsten Aspekte dieser Tests waren:

- **Leichte Bedienbarkeit**: Es wurde überprüft, ob das Design der Benutzeroberfläche leicht verständlich und bedienbar ist.
- **Effizienz**: Dazu wurde die Zeit gemessen, die Benutzer benötigen, um bestimmte Aufgaben im Prototyp zu erledigen.
- **Zufriedenheit**: Nach jedem Usability-Test wurde subjektives Feedback der Benutzer zur allgemeinen Zufriedenheit mit dem GUI-Design gesammelt.

6.6.4 Konzeptionelle Leistungstests

Leistungstests in der konzeptionellen Entwurfsphase wurden durchgeführt, um sicherzustellen, dass die Designkonzepte auch bei späteren Implementierungen großer Datenmengen schnell und reaktionsfähig bleiben. Diese Tests umfassten:

- **Ladezeiten:** Es wurden die erwarteten Ladezeiten basierend auf dem Design überprüft.
- **Reaktionszeit:** Es wurde sichergestellt, dass die geplante Benutzeroberfläche flüssig auf Eingaben reagiert.

6.6.5 Abnahmetests

Abnahmetests mit Stakeholdern und Endbenutzern wurden durchgeführt, um die Akzeptanz der GUI sicherzustellen. Diese Tests umfassten:

- **Review der funktionalen Anforderungen:** Es wurde bestätigt, dass die Anforderungen vollständig und korrekt umgesetzt sind.
- **Feedback zur Benutzerfreundlichkeit:** Es wurde Feedback zur Benutzerfreundlichkeit und mögliche Anpassungen basierend auf den Rückmeldungen gesammelt.

Durch diese umfassenden Tests und Validierungsmaßnahmen wurde sichergestellt, dass die entworfenen Konzepte und entwickelten Prototypen der GUI benutzerfreundlich, effizient und zuverlässig sind. Dies bietet eine solide Grundlage für die weitere Entwicklung der vollständigen GUI, die eine einfache Verwaltung der Schnittstelle zur Integration von Unternehmenssystemen ermöglicht.

7 Ergebnisse

7.1 Möglichkeiten der Generalisierung und deren Auswirkungen

Die Einführung einer generischen und modularen Schnittstelle in der Intralogistik bietet zahlreiche Vorteile. So können Entwicklungsprozesse erheblich beschleunigt und vereinfacht werden. Dies führt zu einer Reduktion der Entwicklungsstunden und damit zu Kosteneinsparungen. Ein wesentlicher Vorteil ist die Wiederverwendbarkeit der Schnittstellenmodule, was den Aufwand für Anpassungen in neuen Kundenprojekten minimiert.

Die generische Schnittstelle ermöglicht es, verschiedene Kommunikationsarten (wie REST, Web-Services und Datenbank-Schnittstellen) zu unterstützen, was die Flexibilität und Anpassungsfähigkeit der Lösung erhöht. Zudem trägt sie zur Konsistenz der Daten bei, indem sie ein einheitliches Datenformat für die Übertragung verwendet. Diese Standardisierung reduziert die Fehleranfälligkeit und sorgt für eine höhere Zuverlässigkeit der Systeme.

Durch die Generalisierung wird zudem die Wartung und Pflege der Schnittstellen vereinfacht. Updates und Erweiterungen können zentral vorgenommen werden, ohne dass jedes Kundenprojekt individuell angepasst werden muss. Dies führt zu einer höheren Effizienz und reduziert die Gefahr von Inkompatibilitäten zwischen den Systemen.

7.2 Erweiterung der Schnittstelle durch moderne Kommunikationsmethoden

Die entwickelte generische Schnittstelle wurde erfolgreich in vier verschiedenen Schnittstellen eingesetzt. Dadurch wurde die Schnittstelle um moderne Kommunikationsmethoden, wie beispielsweise Datenbank-Schnittstellen, Web-Services und REST-APIs, erweitert.

Bei einem Kunden-Projekt kam die Schnittstelle in einer Web-Service-Schnittstelle zum Einsatz. Diese Web-Service-Schnittstelle bietet eine standardisierte Methode, um Daten über das Internet auszutauschen und ermöglicht es verschiedenen Systemen, unabhängig von ihrer Plattform oder Programmiersprache, miteinander zu kommunizieren. Bei einem anderen Kundenprojekt war die Integration von Datenbank-Schnittstellen erforderlich. Diese Integration ermöglicht es, Daten direkt zwischen verschiedenen Systemen über eine Austausch-Datenbank zu übertragen.

Mithilfe dieser Erweiterungen wurde die Kompatibilität und Flexibilität der Schnittstelle genauer untersucht und nachgewiesen, dass sie eine noch größere Flexibilität und Interoperabilität der Systeme ermöglicht.

Diese modernen Kommunikationsmethoden erweitern die Einsatzmöglichkeiten der generischen Schnittstelle und machen sie zu einem leistungsfähigen Werkzeug für die Intralogistik.

7.3 Funktionsfähige GUI zur Schnittstellenadministration

Die Entwicklung der GUI zur Schnittstellenadministration hat gezeigt, dass eine benutzerfreundliche Oberfläche die Verwaltung und Konfiguration der Schnittstellen erheblich erleichtern kann.

Der GUI-Mapper wurde konzeptioniert und entwickelt, um Felder zwischen dem WMS und externen Systemen intuitiv zuzuordnen und zu konfigurieren. Die wesentlichen Funktionen der GUI umfassen die Möglichkeit, das Mapping zwischen Feldern zu konfigurieren, Felder manuell anzulegen, zu bearbeiten und zu löschen, sowie die Konvertierung von Datentypen einzustellen. Die Echtzeit-Validierung und die Möglichkeit, den aktuellen Workspace zu speichern, tragen zur Benutzerfreundlichkeit und Effizienz der Oberfläche bei.

Die GUI soll die Nutzer dabei unterstützen, auch ohne tiefgehende technische Kenntnisse die Schnittstellen zu verwalten. Dies erleichtert es, eine Schnittstelle zu konfigurieren und anzupassen, was zu einer schnelleren Implementierung und einer höheren Flexibilität in Kunden-Projekten führen kann.

8 Diskussion

8.1 Bewertung der entwickelten Lösung

Die entwickelte Lösung zur Generalisierung und Administration von Schnittstellen im WMS zeigt erhebliche Verbesserungen in Bezug auf Effizienz und Flexibilität. Die generische Schnittstelle ermöglicht eine einheitliche Handhabung verschiedener Kommunikationsmethoden wie REST, SOAP und Datenbank-Schnittstellen, wodurch die Wiederverwendbarkeit und Anpassungsfähigkeit erhöht werden. Die GUI wurde konzeptioniert und implementiert, um eine benutzerfreundliche Verwaltung der Schnittstellen zu ermöglichen.

Die ersten Tests haben gezeigt, dass die GUI intuitiv zu bedienen ist und auch von Nutzern ohne tiefgehende technische Kenntnisse verwendet werden kann.

8.2 Wirtschaftliche Bewertung

Die Generalisierung der Schnittstellen und die benutzerfreundliche GUI bieten signifikante wirtschaftliche Vorteile. Durch die Reduktion der Entwicklungsstunden und die Wiederverwendbarkeit der Schnittstellenmodule können Projekte schneller abgeschlossen und Kosten eingespart werden. Die standardisierte Schnittstelle verringert den Aufwand für Anpassungen in neuen Projekten, was zu einer besseren Ressourcennutzung und höheren Effizienz führt. Insgesamt tragen diese Maßnahmen zu einer höheren Wettbewerbsfähigkeit des Unternehmens bei.

8.3 Herausforderungen und Limitationen

Trotz der positiven Ergebnisse gab es auch Herausforderungen und Limitationen. Die Konzeption und Implementierung einer generischen Schnittstelle erforderten eine detaillierte Analyse und Abstraktion bestehender Schnittstellen, was zeitaufwendig war. Die GUI wurde bereits entwickelt, weist jedoch noch erhebliches Weiterentwicklungspotenzial auf. Eine Fortführung der Entwicklung könnte die Funktionalität erweitern und die Anbindung an das System deutlich vereinfachen. Zudem musste die Schnittstelle kontinuierlich gewartet und an neue Technologien angepasst werden, um ihre Relevanz und Effizienz zu erhalten. Eine weitere Herausforderung bestand darin, sicherzustellen, dass die Schnittstelle in verschiedenen Kundenumgebungen problemlos integriert werden kann.

9 Fazit und Ausblick

9.1 Zusammenfassung der wichtigsten Ergebnisse

Diese Bachelorarbeit konzentrierte sich auf die Generalisierung der Schnittstellen im WMS, um einen effizienten Datenaustausch sowie die Echtzeitsynchronisation zu gewährleisten. Diese Generalisierung bildete die notwendige Voraussetzung für die anschließende Konzeption und Entwicklung einer benutzerfreundlichen GUI zur Administration der Schnittstellen zwischen einem WMS und externen Host-Systemen.

Durch die Entwicklung einer generischen Schnittstelle und einer GUI zur Verwaltung von Schnittstellen wurden erhebliche Fortschritte erzielt, die die Konsistenz der Daten verbessern und die Implementierung zukünftiger Schnittstellen beschleunigen.

9.2 Beitrag der Arbeit zur Schnittstellenentwicklung

Ein innovatives Konzept für den Datenaustausch über standardisierte Methoden wie Datenbank-Schnittstellen, Web-Services und REST-APIs wurde entwickelt. Dieses Konzept passt sich den spezifischen Anforderungen der jeweiligen Systeme an.

Die Standardisierung der Datenformate und die Einführung einer generischen Schnittstelle erhöhten die Flexibilität und Anpassungsfähigkeit der Lösung. Diese praxisorientierte Herangehensweise ermöglichte es, reale Herausforderungen in der Systemintegration zu bewältigen und kreative Lösungen zu entwickeln.

Die entwickelte GUI soll eine benutzerfreundliche Verwaltung der Schnittstellen ermöglichen und eine einfache Konfiguration bieten.

9.3 Vorschläge für weiterführende Forschung und Entwicklungen

Der nächste Schritt ist die Weiterentwicklung und weitere Erprobung der konzipierten und entwickelten GUI zur Schnittstellenadministration. Dies wird die Anwendungsmöglichkeiten der Lösung auf unterschiedliche Szenarien und Systeme erweitern.

Die erfolgreiche Umsetzung dieses Projekts hat die Systemintegrationsfähigkeiten gestärkt und die betriebliche Effizienz in der Intralogistik verbessert. Es schafft eine solide Grundlage für zukünftige Anforderungen und Erweiterungen.

Insgesamt zeigt diese Bachelorarbeit, wie generalisierte Schnittstellen die Systemintegration in Unternehmen erheblich verbessern können. Die entwickelte generalisierte Schnittstelle ist eine vollständige Implementierung, die als Basis für die Integration weiterer Schnittstellen verwendet werden kann und die Datenkommunikation in der iFD GmbH zukünftig effizienter gestalten wird.

Quellenverzeichnis

40. XML Schema-based configuration, 2024. In: https://docs.spring.io/spring-framework/docs/4.2.x/spring-framework-reference/html/xsd-configuration.html (08.07.2024)

ASTERA: API First vs. Code First: Der richtige Ansatz für die Produktentwicklung, 2023. In: https://www.astera.com/de/type/blog/api-first-vs-code-first/ (02.07.2024)

COMPONENTSOURCE: Altova MapForce Professional Edition, 2024. In: https://www.componentsource.com/de/product/altova-mapforce-professional (03.07.2024)

COMPUTERWEEKLY.DE: Was ist Simple Object Access Protocol (SOAP)? - Definition von Computer Weekly, 2024 (01.07.2024)

Container Overview :: Spring Framework, 2024. In: https://docs.spring.io/spring-framework/reference/core/beans/basics.html (05.08.2024)

Das weltweit beliebteste API-Testtool | SoapUI, 2024. In: https://www.soapui.org/ (04.07.2024)

Datenmapping-Tools: MapForce, 2024. In: https://www.altova.com/de/mapforce (03.07.2024)

DEINHARD: Was ist Hibernate?, 2024. In: https://www.it-schulungen.com/wir-ueber-uns/wissensblog/was-ist-hibernate.html (04.07.2024)

ESEME, Solomon: GraphQL vs. REST: Alles, was du wissen musst, 2022. In: https://kinsta.com/de/blog/graphql-vs-rest/ (08.08.2024)

Git und Github: Unterschiede einfach erklärt, 2024. In: https://praxistipps.chip.de/git-und-github-unterschiede-einfach-erklaert_110683 (08.07.2024)

GraphQL erklärt, 2024. In: https://www.redhat.com/de/topics/api/what-is-graphql (02.07.2024)

HEINEN ELEKTRONIK GMBH: Was ist GUI? Definition - Graphic User Interface | schnell erklärt, 2023. In: https://heinen-elektronik.de/glossar/gui/ (01.07.2024)

Informationen zu Java 8, 2024. In: https://www.java.com/de/download/help/java8.html (04.07.2024)

JELVIX: Rest vs GraphQL: Comparison, Advantages, and Disadvantages - Jelvix, 2020. In: https://jelvix.com/blog/graphql-vs-rest (02.07.2024)

KASUBKE, Mara: Was ist eine Schnittstelle?, 2020. In: https://1crm-system.de/prozessoptimierung/was-ist-eine-schnittstelle/ (01.07.2024)

MARKINGMYNAME: SQL Server Management Studio (SSMS) - SQL Server Management Studio (SSMS), 2024. In: https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16 (04.07.2024)

MATTHIAS ZEIS: GraphQL: ein wichtiger Schritt für Headless Magento 2, 2022. In: https://www.matthias-zeis.com/magento-2/graphql (04.08.2024)

MOBILELIVE: GraphQL vs SQL: Understanding the Differences and Benefits, 2023. In: https://mobilelive.medium.com/graphql-vs-sql-understanding-the-differences-and-benefits-e29ca31dc0f4 (31.07.2024)

REST APIs Explained - 4 Components, 2024. In: https://mannhowie.com/rest-api (01.07.2024)

RÖHRL, Simona: Integrierte Logistiklösungen: Optimierung durch ERP und WMS, 2024. In: https://mdis-consulting.de/aktuelles/magazin/fach-blogartikel/integrierte-logistikloesungen-optimierung-durch-erp-und-wms.html (27.06.2024)

SAP: Was ist ein WMS (Warehouse-Management-System)? | SAP Insights, 2024. In: https://www.sap.com/germany/products/scm/extended-warehouse-management/what-is-a-wms.html (27.06.2024)

Schnittstellen-Programmierung: Einfache Grundlagen, die begeistern!, 2024. In: https://www.software-mittelstand.info/die-grundlagen-der-schnittstellenprogrammierung-einfach-erklaert/ (02.07.2024)

SEVDESK: ERP-Systeme - Funktion, Kosten & die größten Anbieter im Vergleich, 2024. In: https://sevdesk.de/blog/erp-system/ (27.06.2024)

STEUERWALD, Dana: ERP – Definition des Enterprise Resource Planning, 2017. In: https://www.microtech.de/erp-wiki/erp/ (27.06.2024)

THE INTERACTION DESIGN FOUNDATION: Shneiderman’s Eight Golden Rules Will Help You Design Better Interfaces, 2024. In: https://www.interaction-design.org/literature/article/shneiderman-s-eight-golden-rules-will-help-you-design-better-interfaces (03.07.2024)

WIKIPEDIA: Apache Ant, 2024a. In: https://de.wikipedia.org/w/index.php?title=Apache_Ant&oldid=242085978 (04.07.2024)

WIKIPEDIA: Eclipse (IDE), 2024b. In: https://de.wikipedia.org/w/index.php?title=Eclipse_(IDE)&oldid=244359413 (04.07.2024)

WIKIPEDIA: JavaFX, 2024c. In: https://de.wikipedia.org/w/index.php?title=JavaFX&oldid=243212843 (04.07.2024)

WMS und ERP: Das sind die Unterschiede, 2024. In: https://www.haufe-x360.de/blog/wms-und-erp (27.06.2024)

Anhangsverzeichnis

1. Die Funktion „*process*“ zur Verarbeitung des geladenen Objekts

Diese Funktion zur Verarbeitung des geladenen Objekts wird verwendet, wenn ein neues Objekt in der Tabelle „*databaseWork*“ eingelesen wurde. Folgende Abbildung zeigt die Funktion „*databaseWork*“: **Abbildung 47**

```
    Eine Funktion, die den Identifier aus dem Objekt für Loggingzwecke zwischenspeichert, das
    Telegramm sowie die Kategorie lädt und durch den Aufruf von write- und done-Funktionen
    verarbeitet.

    Params: mgrs – die ManagerFactory vom Server
            pk – eine AtomicReference zur Speicherung des Identifiers für Loggingzwecke
            next – das nächste Objekt

    Throws: Throwable
private void process(final M mgrs, final AtomicReference<P> pk, final T next)
        throws Throwable {

    // Wenn das nächste Objekt null ist, dann nichts mehr tun.
    if (next == null) {
        return;
    }

    // PrimaryKey fürs Logging zwischenspeichern
    pk.set(getIdentifier(next));

    // Telegramm aus dem Objekt lesen
    final String telegram = getTelegram(next);
    // Kategorie aus dem Objekt lesen
    final String category = getCategory(next);

    // Telegramm verarbeiten
    write(mgrs, category, telegram);

    // Dann auf erledigt setzen
    done(mgrs, next);
}
```

Abbildung 46 Die Funktion „*process*“ zur Verarbeitung des geladenen Objekts

2. Die Funktion „*databaseWork*“ für die Verwaltung von Datenbankaktionen

```
    Standard-Implementierung für die Datenbankschnittstellen.
    Hier wird eine bestimmte Tabelle aller waitTime Millisekunden gepoolt

    Author: ZEN, ifd GmbH
protected final void databaseWork() {
    // nun kann auf Telegramme gewartet werden
    while (!Thread.interrupted()) {
        // Solange der Service nicht auf warten steht, Telegramme lesen
        while (getStatus() != ServiceStatus.WAIT) {
            // Platzhalter für die ID des zu verarbeitenden Objects
            final AtomicReference<P> pk = new AtomicReference<>(null);
            // DB-Session öffnen
            final Throwable throwable = openSession(mgrs -> {
                // Nächstes Objekt einlesen
                final T next = read(mgrs);

                // Das neue Objekt verarbeiten
                process(mgrs, pk, next);
            });

            if (throwable instanceof WebServiceException) {
                Log.exc(this, throwable);
                try {
                    Thread.sleep(10_000);
                } catch (final Exception e) {
                    Thread.currentThread().interrupt();
                    Log.exc(this, e);
                    break;
                }
                tryLogError(pk, throwable);
                continue;
            }
            tryLogError(pk, throwable);
            // Wenn nichts gelesen wurde, dann schlafe,
            // sonst wird der Service immer wieder getriggert,
            // es können ja mehrere Telegramme in der Tabelle liegen
            if (pk.get() == null) {
                break;
            }
        }
        try {
            Thread.sleep(this.waitTime);
        } catch (final InterruptedException e) {
            Log.exc(this, e);
            Thread.currentThread().interrupt();
        }
    }
}
```

Abbildung 47 Die Funktion „*databaseWork*“ für die Verwaltung von Datenbankaktionen

3. Die Funktion „*logError*“ zur Behandlung von Fehlermeldungen

```
    Funktion, die den Fehler loggt und am Object anhand dessen Identifier vermerkt.

    Params: mgrs – ManagerFactory
            id – der Identifier des Objects T
            e – Throwable

    Throws: Throwable – Throwables können direkt geworfen werden

    Author: ZEN, ifd GmbH
@Override
protected void logError(final M mgrs, final Long id, final Throwable e)
        throws Throwable {

        final InterfaceProtocolManager ipMgr = mgrs.get(InterfaceProtocolManager.class);

        // Die Exception analysieren und ab 500 Byte abscheiden
        // Nachricht davon laden
        final String errorMsg = StringHelper.cutStringByte(e.toString(), 500);

        // Stacktrace davon laden
        final String thowsInInfo = StringHelper.cutStringByte(Log.getIfdClass(e), 500);

        // Den InterfaceProtocol-Datensatz aus der DB anhand der ID laden
        final InterfaceProtocol ipById = ipMgr.getNotNull(id);

        // Fehler daran setzen
        ipById.setErrorMessage(errorMsg);
        ipById.setErrorThrowsIn(thowsInInfo);
        ipById.setProcessedTime(LocalDateTime.now());

        // wenn es keine Verbindung zum HOST gibt, dann einen entsprechenden Fehlerstatus setzen
        if (e instanceof WebServiceException) {
            ipById.setStatus(InterfaceProtocolStatus.FEHLER_SENDEN);
        } else {
            ipById.setStatus(InterfaceProtocolStatus.FEHLER);
        }

        // Konverter über den Fehler informieren
        this.converter.logError(mgrs, id, ipById.getCategory(), ipById.getTelegram(), e);
}
```

Abbildung 48 Die Funktion "*logError*" zur Behandlung von Fehlermeldungen

4. Die Klasse „*TreeItemData*“ zur Speicherung von Felder-Eigenschaften

```
Die Klasse, welche die Eigenschaften eines Feldes im TreeItem beinhaltet.

Author:  ZEN, ifd GmbH
public class TreeItemData implements Serializable {

    private static final long serialVersionUID = 1L;

    private String              name;
    private Class<?>            type;
    private boolean             nullable;
    private boolean             unique;
    private int                 length;
    private TreeItemData        parent;
    private List<Integer>       mappingIds;
    private List<TreeItemData>  children;
    private boolean             selectedByMappingTable;

    public TreeItemData(
            @JsonProperty("name") final String name,
            @JsonProperty("type") final Class<?> type,
            @JsonProperty("nullable") final boolean nullable,
            @JsonProperty("uniqe") final boolean unique,
            @JsonProperty("length") final int length,
            @JsonProperty("parent") final TreeItemData parent,
            @JsonProperty("children") final List<TreeItemData> children,
            @JsonProperty("mappingIds") final List<Integer> mappingIds) {
        super();
        this.name = name;
        this.type = type;
        this.nullable = nullable;
        this.unique = unique;
        this.length = length;
        this.parent = parent;
        this.children = children != null ? children : new ArrayList<>();
        this.mappingIds = mappingIds;
    }
```

Abbildung 49 Die Klasse „*TreeItemData“* zur Speicherung von Felder-Eigenschaften

Eidesstattliche Erklärung

Ich erkläre an Eides statt,

dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht.

Die Zustimmung des/der beteiligten Unternehmen/s zur Verwendung betrieblicher Unterlagen habe ich eingeholt.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder veröffentlicht noch einer anderen Prüfungsbehörde/-stelle vorgelegt.

Name, Vorname Verfassender

Ort, Datum Abgabetermin

Unterschrift Verfassender

Erklärung zur Prüfung wissenschaftlicher Arbeiten

Die Bewertung wissenschaftlicher Arbeiten erfordert die Prüfung auf Plagiate. Die hierzu von der Staatlichen Studienakademie Glauchau eingesetzte Prüfungskommission nutzt sowohl eigene Software als auch diesbezügliche Leistungen von Drittanbietern. Dies erfolgt gemäß § 7 des Gesetzes zum Schutz der informationellen Selbstbestimmung im Freistaat Sachsen (Sächsisches Datenschutzgesetz - SächsDSG) vom 25. August 2003 (Rechtsbereinigt mit Stand vom 31. Juli 2011) im Sinne einer Datenverarbeitung im Auftrag.

Der Studierende bevollmächtigt die Mitglieder der Prüfungskommission hiermit zur Inanspruchnahme o. g. Dienste. In begründeten Ausnahmefällen kann der Datenschutzbeauftragte der Berufsakademie Sachsen sowohl vom Verfasser der wissenschaftlichen Arbeit als auch von der Prüfungskommission in den Entscheidungsprozess einbezogen werden.

Name: Enabi

Vorname: Mohamad Zakaria

Matrikelnummer: 4004601

Studiengang: Wirtschaftsinformatik

Titel der Arbeit: Analyse und Konzeption einer benutzerfreundlichen GUI zur Vereinfachung der Administration einer Schnittstelle für die Integration von Unternehmenssystemen

Datum: 09.08.2024

Unterschrift:

Abstract zur Bachelorthesis

Studiengang: Wirtschaftsinformatik

Name: Enabi, Mohamad Zakaria

Thema: Analyse und Konzeption einer benutzerfreundlichen GUI zur Vereinfachung der Administration einer Schnittstelle für die Integration von Unternehmenssystemen

Jahr: 2024

Betreuer: Prof. Dr. Mathias Sporer (Staatliche Studienakademie Glauchau)
Ansgar Fischer (iFD GmbH)

Ziel:
Ziel dieser Arbeit ist die Analyse und Konzeption einer benutzerfreundlichen GUI zur einfachen Konfiguration der Schnittstelle und Zuordnung von Sende- und Empfangsfeldern. Darüber hinaus wird die Erweiterung der Schnittstellentechnologie zur Kompatibilität mit modernen Kommunikationsmethoden angestrebt.

Methodik:
Die Arbeit stützt sich auf eine umfassende Literaturrecherche. Die GUI wird mithilfe geeigneter Softwareentwicklungswerkzeuge erstellt. Die Evaluierung der Technologieerweiterung und Generalisierung erfolgt durch Tests mit aktuellen Test- oder Produktivsystemen der Kunden. Die Bewertung der GUI erfolgt durch Endbenutzer, wobei regelmäßige Anpassungen auf deren Feedback basieren.

Ergebnisse:
Als Ergebnis lässt sich festhalten, dass die Entwicklung einer benutzerfreundlichen GUI zur Vereinfachung der Konfiguration durch allgemeine Benutzer einer Schnittstelle machbar ist. Ebenso ist die Erweiterung der Schnittstelle zur Kompatibilität mit modernen Kommunikationsmethoden möglich.

Schlussfolgerung:
Der praktische Lösungsvorschlag zur Entwicklung des GUI-Mappers zur Verwaltung von Schnittstellen sowie die Generalisierung der Schnittstelle können in der Zukunft in der iFD GmbH eingesetzt werden, wobei Anpassungen und möglichen Änderungen nicht ausgeschlossen werden sollten.

Schlüsselwörter:
Schnittstelle, generische Schnittstelle, GUI, GUI-Mapper, Java-Backend, SQL-Datenbank, agile Entwicklung, Daten-Synchronisation, Analyse, Konzeption