HOCHSCHULE
MITTWEIDA
University of Applied Sciences

BACHELORARBEIT

Herr

Edgar Schnurpel

Entwurf und Implementierung einer Importfunktion für XML-Dateien nach dem openTRANS®-Standard

Fakultät
Angewandte Computer- und Biowissenschaften

BACHELORARBEIT

Entwurf und Implementierung einer Importfunktion für XML-Dateien nach dem openTRANS®-Standard

Autor:
Herr
Edgar Schnurpel

Studiengang:
Angewandte Informatik

Seminargruppe:
IF19wS-B

Erstprüfer:
Prof. Dr.-Ing. Uwe Schneider

Zweitprüfer:
Dipl.-Inf. Thomas Haufe

Einreichung:
Leipzig, 26.11.2022

Verteidigung/Bewertung:

Mittweida, 2022

Faculty
Applied Computer Sciences und Biosciences

BACHELOR THESIS

Design and implementation of an import function for XML files in accordance with the openTRANS® standard

author:
Mr.
Edgar Schnurpel

course of studies:
Applied Computer Science

seminar group:
IF19wS-B

first examiner:
Prof. Dr.-Ing. Uwe Schneider

second examiner:
Dipl.-Inf. Thomas Haufe

submission:
Leipzig, 26.11.2022

defence/evaluation:

Mittweida, 2022

Bibliografische Beschreibung:

Schnurpel, Edgar:
Entwurf und Implementierung einer Importfunktion für XML-Dateien nach dem openTRANS®-Standard. 50 Seiten, 19 Abbildungen, Hochschule Mittweida, University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften

Bachelorarbeit, 2022

Referat

Um Geschäftsdokumente wie beispielsweise Aufträge digital zu repräsentieren, werden Standards benötigt, die die darin enthaltenen Informationen einheitlich darstellen. Einer dieser Standards ist openTRANS®.
Diese Arbeit befasst sich mit dem Entwurf und der Implementierung einer Importfunktion für Auftragsdaten, die in Form von openTRANS®-gerechten XML-Dateien vorliegen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

ERP	Enterprise Resource Planning
FTP	File Transfer Protocol
GUI	Graphical User Interface (Grafische Benutzeroberfläche)
ISO	International Organization for Standardization
SQL	Structured Query Language
SSL	Secure Sockets Layer
UML	Unified Modeling Language
XML	Extensible Markup Language

1. Einleitung

1.1 Ingenious Software GmbH

Die Ingenious Software GmbH ist ein Softwareentwicklungsunternehmen mit Sitz in Leipzig. Entwickelt wird „ingenious“, eine ERP-Software für Handwerk und Industrie. Zum Funktionsumfang gehören unter anderem die Verwaltung von Kontakten, Artikeln, Projekten und Bestellungen sowie Terminplanung und Projektzeiterfassung. Die Kunden der Ingenious Software GmbH, zu denen sowohl deutsche als auch ausländische Unternehmen, zum Beispiel aus den USA oder Kanada gehören, sind vor allem in der Rollladen- und Sonnenschutzbranche tätig (Ingenious 2022a, Ingenious 2022b). Die bestehende Software „ingenious“ bildet die Grundlage dieser Arbeit und soll im Rahmen dieser weiterentwickelt werden.

1.2 Zielstellung

Die Digitalisierung hat längst in jeglichen Bereichen unseres Lebens Einzug gefunden. Das gilt auch für den Handel von Waren zwischen Unternehmen – dem Business-to-Business-Bereich. In den damit zusammenhängenden Geschäftsprozessen werden Geschäftsdokument wie Aufträge oder Rechnungen traditionellerweise in Papierform ausgetauscht. Zur Vereinfachung der Prozesse und zur Vermeidung von Papier wurden bereits viele digitale Lösungen geschaffen. Die Software „ingenious“ stellt dafür umfassende Funktionen bereit (Ingenious 2022b).

Fundamental für die Digitalisierung ist die Verwendung von Standards. Sie geben Unternehmen eine gemeinsame Grundlage zur Kommunikation untereinander und bieten damit eine Möglichkeit zum einheitlichen Austausch von Daten (Engels 2017, S. 21). Einer der Standards ist openTRANS®. Er beschreibt Form, Aufbau und Inhalt digitaler Geschäftsdokumente (Schmitz et al. 2009).

Ziel dieser Arbeit ist es, eine Schnittstelle zum openTRANS®-Standard zu entwickeln und diese in die Software „ingenious“ zu integrieren. Mit dieser soll es dann möglich sein, Aufträge in Form von XML-Dateien, die dem openTRANS®-Standard entsprechen, zu importieren.

1.3 Aufbau der Arbeit

Im nachfolgenden 2. Kapitel werden einige grundlegende Themen behandelt, die zum Verständnis dieser Arbeit nötig sind. Das umfasst eine Erklärung, was Geschäftsdokumente sind und wie diese ausgetauscht werden, eine kurze Beschreibung des XML-Dateiformats und eine Vorstellung des openTRANS®-Standards.

Kapitel 3 befasst sich mit dem Prozess der Entwicklung des Softwarekonzepts. Angefangen wird mit der Auswahl eines Vorgehensmodells. Danach erfolgt eine Analyse der Anforderungen, mit dessen Ergebnissen die grafische Benutzeroberfläche erstellt und die Softwarearchitektur erarbeitet wird.

Um einen tieferen Einblick in das entwickelte Software-Modul zu geben, werden in Kapitel 4 einzelne, ausgewählte Details zur konkreten Implementierung vorgestellt.

Kapitel 5 bildet mit einer Zusammenfassung und dem Ausblick den Abschluss dieser Arbeit.

2. Grundlagen

2.1 Geschäftsdokumente

Kelkar definiert Geschäftsdokumente „als strukturierte und in sich abgeschlossene Informationspakete [...], wie sie zwischen Unternehmen im Rahmen ihrer Geschäftstätigkeit ausgetauscht werden“ (Jessen et al. 2003, S. 175).
Die im Rahmen dieser Arbeit relevanten Geschäftsdokumente sind der Auftrag und die Auftragsbestätigung.

Abbildung 1: Austausch von Geschäftsdokumenten
Quelle: Eigene Darstellung

Abbildung 1 zeigt, wie diese zwischen Unternehmen ausgetauscht werden. Die Interaktion beginnt mit dem Aufgeben einer Bestellung eines Kunden an einen Lieferanten (Auftrag). Der Lieferant sendet eine Auftragsbestätigung, sobald er den Auftrag bearbeitet. Diese enthält alle Artikel, die auch im Auftrag enthalten sind. Sollten Daten im Vergleich zum Auftrag abweichen, zum Beispiel in Form höherer Preise, dann ist der Einkäufer dafür verantwortlich, dies zu prüfen und bei Bedarf zu reagieren.
Eine genauere Beschreibung dazu, welche Informationen in Auftrag und Auftragsbestätigung enthalten sind, erfolgt im Rahmen der Vorstellung des openTRANS®-Standards in Kapitel 2.3.

2.2 XML

Extensible Markup Language, kurz XML, ist ein vom World Wide Web Consortium (W3C) entwickeltes, textbasiertes Format für die Repräsentation strukturierter Daten. XML wird für den Austausch von Daten zwischen Programmen auf lokaler Ebene sowie über Netzwerke genutzt (W3C 2015).
Die grundlegendsten Bestandteile sind Elemente und Attribute. Folgendes Beispiel soll die Struktur dieser Bestandteile innerhalb einer XML-Datei aufzeigen:

```
<element1 attribut1="Attribut">
      <element2>Element 2</element2>
      <element3> Element 3</element3>
</element1>
```

XML wurde mit Hinblick auf folgende Ziele entwickelt:

- Einfache Nutzbarkeit über das Internet
- Unterstützung einer Vielzahl von Anwendungen
- leichte Entwicklung von Programmen, die XML-Dokumente verarbeiten
- gute Lesbarkeit für Menschen
- einfache Erstellung von XML-Dokumenten
- Bündigkeit ist von minimaler Bedeutung

(W3C 2008).

Auch der openTRANS®-Standard nutzt XML als Austauschformat. Genaueres dazu wird im folgenden Kapitel erläutert.

2.3 Der openTRANS®-Standard

Der openTRANS®-Standard wurde basierend auf dem Standard „BMEcat®“ vom Fraunhofer Institut und der Universität Duisburg-Essen entwickelt. Darin wird der Aufbau und der Inhalt verschiedener Geschäftsdokumente, welche im Business-to-Business-Bereich genutzt werden, spezifiziert. Folgende werden im Standard beschrieben:

- Angebotsanforderung
- Angebot
- Auftrag
- Auftragsänderung
- Auftragsbestätigung
- Lieferavis
- Wareneingangsbestätigung
- Rechnung
- Rechnungsliste
- Zahlungsavis

Diese zehn Geschäftsdokumente sollen lediglich einen Rahmen für die Interaktion von Geschäftspartnern vorgeben. In welcher Art und Weise sie genutzt werden, wird vom openTRANS®-Standard nicht vorgeschrieben (Schmitz et al. 2009a, S. 7).

In dieser Arbeit die beiden Teilbereiche Auftrag (ORDER) und Auftragsbestätigung (ORDERRESPONSE) betrachtet.

Um die Geschäftsdokumente zu kodieren, wird XML genutzt. Dies ermöglicht eine strukturierte Darstellung von Daten. Die Bestandteile eines Dokuments im openTRANS®-Standard werden genauso wie in XML bezeichnet, es existieren also „Elemente“ und „Attribute“. Zusätzlich zu diesen Begriffen werden in openTRANS® noch weitere Strukturen definiert, die zur Beschreibung der Elemente und deren Beziehung untereinander dienen. Das sind eine Kardinalität, die Sequenz, die Auswahl sowie Muss- und Kann-Felder. Alle Elemente, Sequenzen und Auswahlen sind entweder vom Typ „muss“ oder vom Typ „kann“. (ebd., S. 7 - 9).

Tabelle 1 erklärt eben genannte Strukturen:

Name der Struktur		**Bedeutung**
Kardinalität	Muss-Element	Muss entweder ein Mal oder ein bis n-Mal vorkommen.
	Kann-Element	Kann entweder ein Mal oder bis zu n-Mal vorkommen.
Sequenz	Muss-Sequenz	Menge von Elementen, von denen alle vorhanden sein und in einer vorgegebenen Reihenfolge vorkommen müssen.
	Kann-Sequenz	Menge von Elementen, die vorhanden sein können, und in einer vorgegebenen Reihenfolge vorkommen müssen.
Auswahl	Muss-Auswahl	Menge von Elementen, von denen genau eines vorkommen muss.
	Kann-Auswahl	Menge von Elementen, von denen höchstens eines vorkommen kann. Darf auch gar nicht vorhanden sein.

Tabelle 1: Strukturen, die openTRANS-Elemente näher beschreiben.
Quelle: in Anlehnung an (Schmidt et al. 2009a, S. 9 – 16)

Eine XML-Datei ist openTRANS®-konform, wenn alle Muss-Felder vorhanden sind und sie kein Element enthält, welches nicht durch eines der im Standard spezifizierten Kann-Felder beschrieben wird. Außerdem müssen alle Elemente in der angegebenen Reihenfolge vorkommen und die vorgegebene Kardinalität besitzen (ebd., S. 11).

Im Folgenden soll der grundlegende Aufbau eines openTRANS®-Dokuments anhand der ORDER erklärt werden.
Der Name des Wurzelelements ist immer die englische Bezeichnung des Geschäftsdokuments, in diesem Falle also ORDER. Das Wurzelelement besitzt drei Kindelemente, die die drei Hauptbestandteile eines Auftrags repräsentieren. Diese sind:

- ORDER_HEADER
- ORDER_ITEM_LIST
- ORDER_SUMMARY

Im ORDER_HEADER werden grundlegende Informationen zum Auftrag hinterlegt. Das umfasst die Auftragsnummer, den Zeitstempel des Auftrags, das Lieferdatum, die daran beteiligten Geschäftspartner und vieles mehr.

Die ORDER_ITEM_LIST enthält alle Positionen des Auftrags. Zu jeder Position gehört mindestens eine Positionsnummer (beispielsweise „1“ für die erste Position, „2“ für die zweite etc.), die eindeutige Identifikationsnummer des Artikels, die Bestellmenge und die Einheit, in der der Artikel bestellt wird, zum Beispiel „Stück“. Optional kann eine Position unter anderem durch Produktmerkmale oder Produktkomponenten beschrieben werden. Auch der Preis des Artikels kann hier festgelegt werden.

Die ORDER_SUMMARY besitzt zwei Kindelemente, die redundante Informationen enthalten. Das erste ist die Anzahl der im Auftrag enthaltenen Positionszeilen, das zweite die Gesamtsumme in Form des Bruttopreises aller Positionen. Weil diese Informationen schon durch die ORDER_ITEM_LIST beschrieben werden, sind sie lediglich für Kontrollzwecke vorgesehen (ebd, S. 16 – 197).

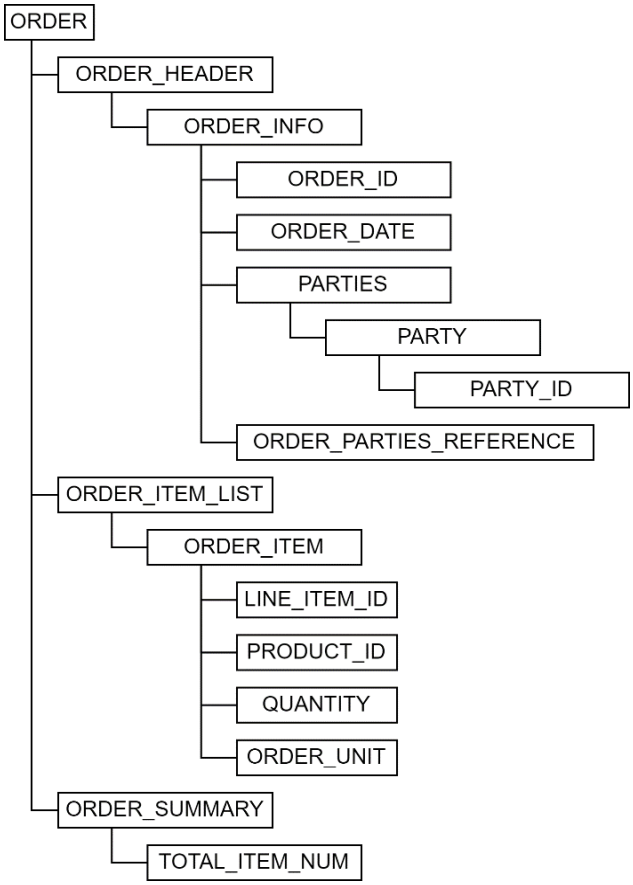


Abbildung 2: Aufbau einer ORDER
Quelle: in Anlehnung an (Schmitz et al. 2009a, S. 224 – 237)

In Abbildung 2 wird der oben beschriebene Aufbau in gekürzter Form dargestellt. Es sind nur Elemente enthalten, die zwingend in einer XML-Datei vorhanden sein müssen, damit sie openTRANS®-konform ist.

Die ORDERRESPONSE ist grundlegend genauso wie die ORDER aufgebaut, auch die meisten Elemente sind identisch. Einige Elemente, die in der ORDER enthalten sind, fehlen jedoch in der ORDERRESPONSE. Das sind zum Beispiel Informationen zur Zahlungsweise oder Informationen zum Transport der Ware. Im Gegensatz dazu enthält die ORDERRESPONSE einige für die Auftragsbestätigung spezifische Elemente, zum Beispiel die SUPPLIER_ORDER_ID, welches die Auftragsnummer des Lieferanten beschreibt (Schmitz et al. 2009a, Schmitz et al. 2009b).

3. Softwaretechnisches Konzept

Das softwaretechnische Konzept beinhaltet die Erfassung aller für die Entwicklung einer Software notwendigen Voraussetzungen, Gedanken und Entwürfe, die zur Implementierung genutzt werden. In diesem Kapitel wird der Prozess der Erstellung des Konzepts beschrieben. Er verläuft über die Auswahl eines Vorgehensmodells über das Requirements Engineering bis hin zur Planung der Software-Architektur (Balzert 2009).

3.1 Auswahl eines Vorgehensmodells

„Ein Vorgehensmodell enthält die zur Erreichung eines bestimmten Ergebnisses notwendigen Aktivitäten, und stellt diese in einer sachlogischen Reihenfolge dar.“ (Alpar et al. 2019, S. 351).
Da sich laut dieser Definition sowohl die Schritte zur Entwicklung einer Software als auch der Zeitpunkt ihrer Abarbeitung je nach Vorgehensmodell unterscheiden, ist es wichtig, die Entscheidung nach welchem Modell vorgegangen werden soll, vor dem eigentlichen Entwurfs- und Entwicklungsprozess zu treffen. Im Folgenden werden drei Modelle vorgestellt und anschließend anhand von Vor- und Nachteilen auf ihre Eignung für die Aufgabe geprüft.

3.1.1 Wasserfall-Modell

Im Wasserfall-Modell durchläuft die Entwicklung vorgegebene Phasen, die vollständig und sequenziell abgearbeitet werden müssen.
Im klassischen Wasserfall-Modell gibt es folgende fünf Phasen:

- Analyse und Anforderungserhebung
- Architekturentwurf
- Implementierung
- Verifikation und Integration
- Betrieb und Wartung

(Broy; Kuhrmann 2021, S. 86f.)

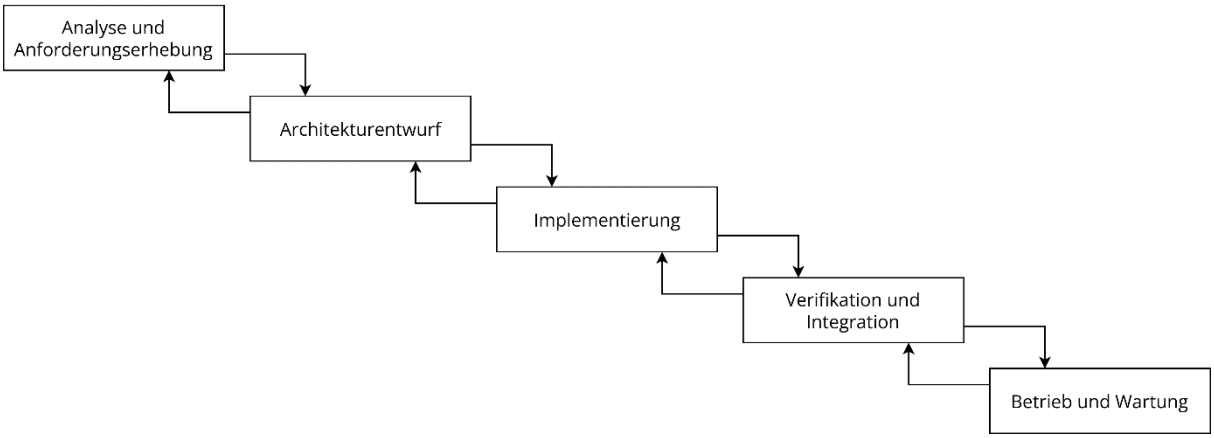


Abbildung 3: klassisches Wasserfall-Modell
Quelle: in Anlehnung an (Broy; Kuhrmann 2021, S. 87)

Wie in Abbildung 3 zu erkennen ist, wird die sequenzielle Natur dieses Modells dadurch aufgeweicht, dass Rücksprünge zu einer vorherigen Phase ermöglicht werden. Dies sollte jedoch nur zur Korrektur fehlerhafter Resultate genutzt werden (Sandhaus et al. 2014, S. 29).

3.1.2 V-Modell

Das V-Modell ist eine Erweiterung des Wasserfall-Modells. Dabei wird jede Phase des Wasserfall-Modells um qualitätssichernde Maßnahmen erweitert (Sandhaus et al. 2014, S. 31). Das „V“ im Namen des Modells bezieht sich einerseits auf die Form der Darstellung (siehe Abbildung 4), welche den Buchstaben V nachbildet, andererseits auf die Begriffe Verifikation und Validation. Verifikation beschreibt die Prüfung, ob die Implementierung der Spezifikation entspricht, ohne dabei auf die Qualität der Spezifikation zu achten, wohingegen bei der Validation auf die Tauglichkeit des Endprodukts für den Kunden geachtet wird (Hoffmann 2013, S. 497).

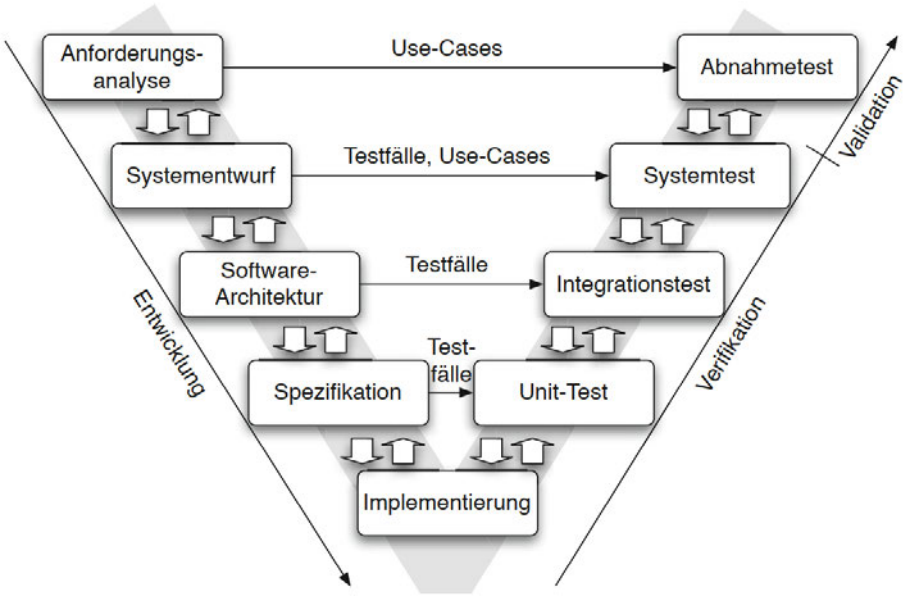


Abbildung 4: V-Modell
Quelle: Hoffmann 2013, S. 497

In Abbildung 4 ist die für dieses Modell typische V-Form zu erkennen. Außerdem wird die Bedeutung der Qualitätssicherung deutlich: aus jedem Entwicklungsschritt auf der linken Seite werden Erkenntnisse abgeleitet, die in den Tests auf der rechten Seite zur Anwendung kommen.

3.1.3 Scrum

Scrum ist ein Vertreter der agilen Softwareentwicklung. Diese hat das Ziel, Software inkrementell und in engem Kundenkontakt zu entwickeln (Alpar et al. 2019, S. 371). Scrum eignet sich für komplexe Projekte, welche sich nicht im Detail planen lassen.
In Scrum werden drei Rollen definiert, welche in Tabelle 2 zu sehen sind:

Rolle	**Aufgaben**
Product Owner	• verantwortlich für das Ergebnis • legt Anforderungen fest, die in einem Product Backlog festgehalten und priorisiert werden
Entwicklungsteam	• legt Sprint Backlog an, welcher eine Auswahl von Elementen des Product Backlogs enthält • implementiert Elemente aus Sprint Backlog innerhalb einer Iteration (Sprint; maximal 30 Tage) • Durchführung von Daily Scrums zur Besprechung des Fortschritts
Scrum Master	• nicht Teil des Entwicklungsteams • verantwortlich für die Einhaltung des Scrum-Prozesses • Mentoren-Rolle

Tabelle 2: Rollen in Scrum
Quelle: in Anlehnung an (Broy; Kuhrmann 2021, S. 104 ff.)

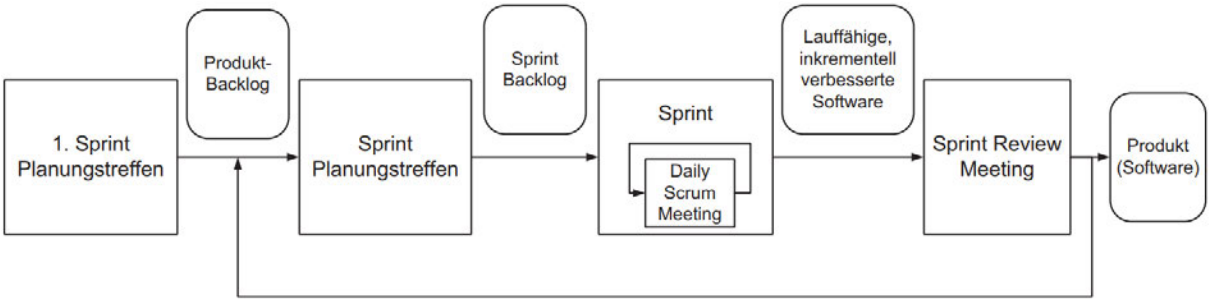


Abbildung 5: Ablauf von Scrum
Quelle: Alpar et al. 2019, S. 372

Abbildung 5 zeigt die Schritte, die bei Scrum durchlaufen werden. Zunächst wird ein erstes Planungstreffen abgehalten, in dem erste Anforderungen im Product Backlog festgehalten werden. Danach beginnt ein Sprint, also eine Iteration in der Entwicklung. Dafür werden vom Entwicklungsteam Elemente aus dem Product Backlog ausgesucht, in einen Sprint Backlog geschrieben und bis zum Ende des Sprints implementiert. Zusätzlich finden

täglich Meetings statt, in denen Fortschritte und Probleme besprochen werden. Am Ende eines Sprints sollte eine lauffähige Software vorliegen, die zusammen mit dem Kunden im Sprint Review Meeting diskutiert wird. Auf Grundlage dieses Meetings können neue Anforderungen in das Product Backlog aufgenommen werden. Anschließend kann ein neuer Sprint beginnen (Broy; Kuhrmann 2021, S. 106).

3.1.4 Vergleich der Vorgehensmodelle

Im Folgenden werden Vor- und Nachteile der drei vorgestellten Vorgehensmodelle aufgeführt. Anschließend erfolgt eine kurze Bewertung hinsichtlich der Eignung jedes Modells für die Aufgabe.
Bei den in Tabelle 3 aufgeführten Punkten handelt es sich teilweise um für die im Rahmen der Aufgabe und dieser Arbeit ausgewählte Vor- und Nachteile. Bei anderen Softwareentwicklungs-Projekten können sich aus den Eigenschaften der einzelnen Vorgehensmodelle durchaus andere Resultate ergeben.

Modell	**Vorteile**	**Nachteile**
Wasserfall-Modell	• einfache und übersichtliche Organisation des Entwicklungsprozesses • Eignung für Projekte, in denen die Anforderungen und die gewünschte Lösung klar sind • formale Spezifikation	• Änderungen an Anforderungen werden nicht berücksichtigt • Gefahr, dass Planungs- und Entwicklungsfehler zu spät bemerkt werden • zeitlich parallele Ausführung von Aktivitäten nur eingeschränkt möglich
V-Modell	• enge Bindung zwischen Entwicklung und Qualitätssicherung • Ableitung der Testfälle über gesamte Entwicklung hinweg • formale Spezifikation	• stark erhöhter Aufwand durch umfassende Qualitätssicherung
Scrum	• Flexibilität durch iteratives Vorgehen: auf sich ändernde Anforderungen kann rechtzeitig reagiert werden • lauffähige Version der Software nach jeder Iteration	• idealerweise mindestens drei Entwickler • Projekt hat kein definiertes Ende • durch Fokus auf Ergebnisse fehlt formale Spezifikation

Tabelle 3: Vergleich von Wasserfall-Modell, V-Modell und SCRUM
Quelle: in Anlehnung an (Broy; Kuhrmann 2021, S. 88 – 106), (Sandhaus et al. 2014, S. 42 – 50), (Hoffmann 2013, S. 497f.)

Scrum bietet zwar eine hohe Flexibilität, aber dadurch, dass eine Änderung der Anforderungen bei dieser Aufgabe unwahrscheinlich ist, hat dieser Vorteil keine große Relevanz. Viel schwerer fällt ins Gewicht, dass bei Scrum keine formale Spezifikation vorgesehen ist,

was eine Benutzung im Rahmen dieser Arbeit ausschließt, da diese die Arbeitsgrundlage eines solchen Projektes bei der Ingenious Software GmbH bildet.

Eine so intensive Qualitätssicherung wie im V-Modell ist einerseits von der Ingenious Software GmbH nicht vorgesehen, andererseits würde das Betrachten dieses Aspekts den Rahmen dieser Arbeit sprengen. Deshalb ist auch dieses Modell nicht geeignet.

Das Wasserfall-Modell bringt durch sein schrittweises Abarbeiten einzelner Phasen eine gewisse Planungssicherheit mit sich. In Zusammenhang mit der Annahme, dass sich die Anforderungen nicht ändern, bietet das Modell eine gute Struktur für Arbeiten wie diese. Deshalb fällt die Wahl für ein Vorgehensmodell auf das Wasserfall-Modell. Zwar wird nicht auf alle Phasen des Modells explizit eingegangen, der zugrundeliegende Ablauf behält trotzdem seine Relevanz.

3.2 Requirements Engineering

Dieses Kapitel befasst sich mit der ersten Phase des Wasserfall-Modells, der Analyse und Anforderungserhebung.

3.2.1 Pflichtenheft

Für diesen Schritt eignet sich ein Pflichtenheft. Balzert definiert ein Pflichtenheft als „Anforderungsdokument, das die Anforderungen an ein neues Produkt aus Auftraggeber- und Auftragnehmer-Sicht festlegt.“ (Balzert 2009, S. 597). Zudem soll ein Pflichtenheft nicht beschreiben, wie Anforderungen umgesetzt werden, sondern sich darauf beschränken, welche Anforderungen gestellt werden. Das „Wie“ wird erst im fachlichen Feinkonzept beschrieben. Es muss außerdem in einem Detailgrad verfasst sein, der nach der Entwicklung der Software eine Kontrolle auf Vollständigkeit der in ihr erbrachten Leistungen ermöglicht (2003, S. 10f.). Die folgenden sechs Unterkapitel stellen die Ergebnisse der Anforderungsanalyse und damit die Beschreibung der Anforderungen in Form eines Pflichtenhefts dar.

3.2.1.1 Visionen

/V10/ Die Software „ingenious“ soll durch die Erweiterung um eine Komponente „openTRANS®-Import“ (nachfolgend nur als „Komponente“ bezeichnet) in die Lage versetzt werden, XML-Dateien, die dem openTRANS®-Standard entsprechen, zu importieren.

3.2.1.2 Rahmenbedingungen

/R10/ „ingenious“ ist eine ERP-Software für Windows.

/R20/ Zielgruppe sind die Nutzer der Software „ingenious“.

3.2.1.3 Funktionale Anforderungen

/F10/ Die Komponente muss dem Nutzer die Möglichkeit bieten, openTRANS®-konforme XML-Dateien, die zum Teilbereich „ORDER“ gehören, zu importieren.

/F11/ Der Nutzer soll als Antwort auf den Eingang einer ORDER eine openTRANS®-konforme XML-Datei des Teilbereichs „ORDERRESPONSE“ an einen Kunden zurückschicken können.

/F12/ Die Auftragsdaten können sich auf einzelne Artikel und/oder auf Stücklisten-Artikel beziehen. Beide müssen importiert werden können.

/F20/ Die Komponente muss dem Nutzer die Möglichkeit bieten, für jeden seiner Kontakte separat Einstellungen für den Import vorzunehmen.

/F21/ Zu den Einstellungsmöglichkeiten gehört die Wahl, ob Dateien entweder von einem Verzeichnis auf dem Computer des Nutzers oder von einem Verzeichnis auf einem FTP-Server importiert werden sollen.

/F22/ Zu den Einstellungsmöglichkeiten gehört, dass verschiedenartige Fehlermeldungen an eine angegebene E-Mail-Adresse verschickt werden können.

/F23/ Die Einstellungen müssen permanent in der Datenbank gespeichert und nach dem Öffnen eines Kontaktes wieder geladen werden.

/F30/ Die Komponente muss dem Nutzer die Möglichkeit bieten, Artikeldaten aus einer openTRANS®-konformen XML-Datei in eine Stückliste einzupflegen.

/F31/ Die Komponente muss dafür dem Nutzer die Möglichkeit bieten, eine Konfiguration einer Stückliste für einen Stücklisten-Artikel anzulegen, zu modifizieren und zu löschen.

/F32/ Für die Konfiguration muss die Komponente dem Nutzer die Möglichkeit bieten, jede den Stücklisten-Artikel beschreibende Information aus der openTRANS®-konformen XML-Datei einer oder mehreren Eigenschaften von Bauteilen einer ingenious-internen Stückliste zuzuordnen.

/F33/ Die Komponente muss dem Nutzer die Möglichkeit bieten, eine Zuordnung mit Bedingung vorzunehmen. Sie soll bestimmen, unter welchen Voraussetzungen ein Stücklisten-Element einen angegebenen Wert annimmt.

/F34/ Die Komponente muss dem Nutzer die Möglichkeit bieten, die Konfiguration für eine oder mehrere Varianten des Stücklisten-Artikels vorzunehmen.

/F35/ Die Komponente soll dem Nutzer die Möglichkeit bieten, eine Konfiguration eines Stücklisten-Artikels zu duplizieren, sodass sie anschließend an einen anderen Stücklisten-Artikel angepasst werden kann.

/F36/ Die Komponente soll dem Nutzer die Möglichkeit bieten, ein oder mehrere Elemente einer Konfiguration eines Stücklisten-Artikels zu kopieren und bei der Konfiguration eines anderen Stücklisten-Artikels wieder einzufügen.

/F40/ Die Komponente muss dem Nutzer die Möglichkeit bieten, zu wählen, in welches Datenbankfeld eine Information aus einer openTRANS®-konformen XML-Datei gespeichert werden soll.

/F50/ Die Komponente muss dem Nutzer die Möglichkeit bieten, zu wählen, welche der durch den openTRANS®-Standard beschriebenen Elemente importiert werden sollen.

/F60/ Die Komponente muss in „ingenious“ für jede importierte Datei mit den darin enthaltenen Auftragsdaten eine Auftragsbestätigung anlegen und in der Datenbank speichern.

/F61/ Wenn in der XML-Datei Stücklisten-Artikel beschrieben werden, dann müssen diese entsprechend der Konfiguration aus /F30/ bis /F34/ in der Auftragsbestätigung berücksichtigt werden.

/F70/ Zwei Benutzerrechte sollen den Zugriff auf die beschriebenen Einstellungsmöglichkeiten regeln. Eines ermöglicht es dem Nutzer, die Einstellungsmöglichkeiten einzusehen, das andere ermöglicht ein Einsehen und zusätzlich eine Bearbeitung aller Einstellungen.

/F80/ Die Komponente soll den Import per Scheduler durchführen können, das heißt automatisch und zeitgesteuert mit den im „ingenious“ dafür vorgesehenen Funktionen.

3.2.1.4 Abnahmekriterien

/A10/ Gültiges Abnahmeszenario: Eine Konfiguration neu erstellen, Stücklisten-Artikel konfigurieren, Import starten, Überprüfung der Vollständigkeit der erstellten Auftragsbestätigungen.

3.2.1.5 Glossar

Artikel	Kann ein einzelnes Produkt sein oder ein sogenannter Stücklisten-Artikel, also ein Artikel, der eine Stückliste besitzt.
Konfiguration	Gesamtheit aller Einstellungen, die in einer Stückliste für einen Stücklisten-Artikel vorgenommen wurden. Beschreibt auch den Vorgang des Konfigurierens.
Kontakt	Als Kontakt wird in „ingenious“ ein mit dem Nutzer der Software in Beziehung stehendes Unternehmen bezeichnet. Das kann beispielsweise ein Kunde sein.
Stückliste	Eine Stückliste ist eine Liste von konfigurierbaren Bauteilen und sonstigen Eigenschaften. Das Vorhandensein von Bauteilen und Eigenschaften kann abhängig oder unabhängig von der Existenz anderer sein. Durch verschiedene Ausprägungen von Bauteilen mit unterschiedlichen Abhängigkeiten ergeben sich viele Kombinationen, die aber als Ganzes betrachtet einen kompletten Artikel mit einheitlichem Namen - einen Stücklisten-Artikel - ergeben.

3.2.1.6 Qualitätsanforderungen

Neben den funktionalen Anforderungen werden im Pflichtenheft noch Qualitätsanforderungen festgelegt (Balzert 2009, S. 466). Die ISO beschreibt acht Haupt-Qualitätsanforderungen an Software, welche in Tabelle 4 aufgelistet werden:

Qualitätsanforderung	**Bedeutung**
Funktionalität	beschreibt, in welchem Maß die Software unter spezifizierten Bedingungen angegebene und implizite Anforderungen erfüllt
Effizienz	beschreibt die Leistung relativ zur Menge der genutzten Ressourcen unter spezifizierten Bedingungen
Kompatibilität	beschreibt, in welchem Maß die Software Informationen mit anderen Software-Produkten, -systemen oder -komponenten austauschen kann oder ihre erforderlichen Aufgaben erfüllt, während sie sich eine Hard-/Softwareumgebung teilen
Benutzerfreundlichkeit	beschreibt, in welchem Maß die Software von ihren Nutzern verwendet werden kann, um bestimmte Ziele effektiv und effizient zu erreichen
Zuverlässigkeit	beschreibt, in welchem Maß die Software unter vorgegebenen Bedingungen bestimmte Funktionen für eine vorgegebene Zeitspanne erfüllen kann
Sicherheit	beschreibt, in welchem Maß die Software Informationen und Daten schützt, sodass Personen oder andere Software nur so viel Zugriff darauf haben, wie es ihre Befugnis erlaubt
Wartbarkeit	beschreibt, wie effektiv und effizient eine Software verbessert, modifiziert oder korrigiert werden kann, um auf Änderungen in Umgebung und Anforderungen zu reagieren
Übertragbarkeit	beschreibt, wie gut die Software von einer Hardware- oder Softwareumgebung in eine andere transferiert werden kann

Tabelle 4: Qualitätsanforderungen für Software laut ISO
Quelle: in Anlehnung an (ISO/IEC 2011)

Qualitätsanforderung	sehr gut	gut	normal	nicht relevant
Funktionalität	X			
Effizienz			X	
Kompatibilität			X	
Benutzerfreundlichkeit		X		
Zuverlässigkeit		X		
Sicherheit			X	
Wartbarkeit		X		
Übertragbarkeit				X

Tabelle 5: Qualitätsanforderungen für die Komponente
Quelle: in Anlehnung an (ISO/IEC 2011) und (Balzert 2009, S. 495f.)

Tabelle 5 zeigt die für die Komponente gewählten Qualitätsanforderungen. Eine hohe Funktionalität ist besonders wichtig, da es sich bei den zu verarbeitenden Daten um Auftragsdaten handelt, welche bei nicht korrekter Verarbeitung von „ingenious“ falsch erfasst würden und damit möglicherweise finanzielle Schäden mit sich bringen.
An Effizienz und Kompatibilität werden keine besonderen Anforderungen gestellt, sollten sich jedoch am Rest der ingenious-Software orientieren und in einem normalen Rahmen bleiben. Die Benutzerfreundlichkeit sollte gut sein. Das umfasst beispielsweise, dass alle Einstellungsmöglichkeiten, die von der Komponente bereitgestellt werden, zentral von einer Stelle aus erreichbar sein sollten.
Die Zuverlässigkeit schließt sich an die oben beschriebene Situation zur Funktionalität an. Da der Prozess des Imports aber immer weitgehend identisch ist und daher keine größeren Fehlerquellen aufweist, wurde die Zuverlässigkeit als gut eingestuft. Die Komponente umfasst keine sicherheitskritischen Vorgänge, die nicht schon vom Rest der ingenious-Software abgedeckt wäre, weshalb der Punkt Sicherheit mit normal bewertet wurde.
Die Wartbarkeit der Komponente sollte gut sein, unter anderem weil zu einem späteren Zeitpunkt auch der Import von Auftragsänderungen, welche auch durch den openTRANS®-Standard spezifiziert werden, vorgesehen ist. Dieser sollte sich leicht in das vorhandene System integrieren lassen. Da die Komponente nur eine von vielen Bestandteilen von „ingenious“ ist, wird keine für die Komponente spezifische Anforderung an die Übertragbarkeit gestellt und wird damit als nicht relevant eingestuft.

3.2.2 Fachliche Lösung

Nachdem die Anforderungen spezifiziert wurden, kann darauf aufbauend die fachliche Lösung erstellt werden. Sie ist das wichtigste Ergebnis des Requirement Engineerings. Die fachliche Lösung muss die Anforderungen aus dem Pflichtenheft korrekt und vollständig umsetzen und dabei eine präzise Formulierung verwenden. Dabei wird ein Modell erstellt, welches die zu entwickelnde Software beschreibt (Balzert 2009, S. 547, 597).

3.2.2.1 Use Cases

Ein Teil der fachlichen Lösung ist die Erstellung von Use Cases (dt. Anwendungsfall). In den Use Cases erfolgt eine Betrachtung des Systems von außen, daher ist die Frage, wie die Anforderungen umgesetzt werden, z. B. wie die Benutzeroberfläche aussieht, dafür irrelevant. Es sollen lediglich die Bedürfnisse von Akteuren, also Nutzern der Software abgebildet werden (Balzert 2009, S. 251f.).

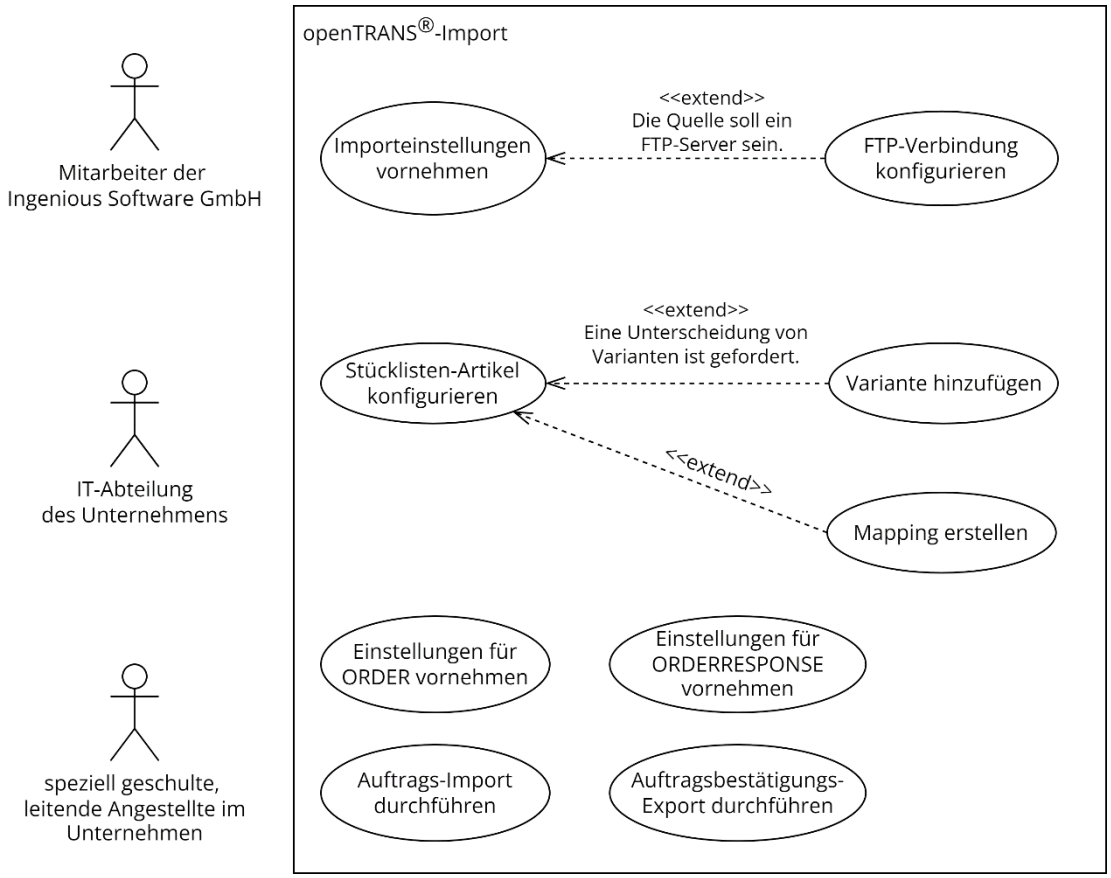


Abbildung 6: Use-Case-Übersicht
Quelle: Eigene Darstellung

Abbildung 6 zeigt eine Übersicht aller Use Cases der Komponente „openTRANS®-Import“. Abweichend von üblichen Use Case-Diagrammen wurden hier Verbindungen von den Akteuren zu den Use Cases weggelassen, damit die Übersichtlichkeit der Darstellung erhalten bleibt. Grundsätzlich kann jeder Use Case von jedem Akteur durchgeführt werden. Die unteren beiden Akteure sind die Nutzer von „ingenious“, in diesem Fall also die Empfänger des Auftrags.

Durch den Use Case „Importeinstellungen vornehmen“ wird beschrieben, welche allgemeinen Einstellungen ein Nutzer für einen bestimmten Kontakt tätigen muss, bevor er den Import startet. Das umfasst optional das Festlegen einer E-Mail-Adresse, an die Fehlermeldungen geschickt werden, sollten welche während des Importprozesses auftreten. Hier stehen dem Nutzer zwei Arten von Fehlermeldungen – Stücklistenwarnungen oder Stücklistenfehler – zur Verfügung. Außerdem legt der Nutzer fest, wo sich der Ordner für die Quelldateien befindet, wo Dateien, für die der Import fehlschlägt, abgelegt werden und wo Dateien, deren Import erfolgreich verlaufen ist, gespeichert werden. Für jeden dieser Ordner muss der Nutzer die Auswahl treffen, ob diese sich auf einem Netzwerkverzeichnis oder auf einem FTP-Server befinden.
Dieser Use Case wird durch den Use Case „FTP-Verbindung konfigurieren“ erweitert. Er tritt ein, wenn sich der Nutzer dazu entschieden hat, dass mindestens einer der oben genannten Ordner auf einem FTP-Server liegt. Dann kann der Nutzer Host, Nutzername und Passwort bestimmen, sowie ob zur Übertragung SSL genutzt wird.

Der Use Case „Stücklisten-Artikel konfigurieren“ beschreibt den Prozess des Erstellens einer Konfiguration und deren Bearbeitung. Für die Erstellung einer Konfiguration muss der Nutzer mindestens den Namen des Artikels und einen firmeninternen Bezeichner angeben, um den Artikel in der XML-Datei eindeutig zu identifizieren.

Der Use Case „Mapping erstellen“ stellt den Hauptteil des Konfigurierens dar: das Anlegen eines Mappings, also einer Zuweisung von Produkteigenschaften in der internen Darstellung des Datei-Lieferanten zu einer ingenious-internen Beschreibung. Zum Beispiel könnte eine Eigenschaft, welche im Unternehmen des Datei-Lieferanten mit „OT1“ bezeichnet wird und den Wert „Z1“ hat, auf den ingenious-Bezeichner „Putzträgerplatte“ mit

dem Wert „außen“ gemappt werden. Mit einem korrekten und vollständigen Mapping wird sichergestellt, dass der Auftrag, der importiert werden soll, dann auch genauso in „ingenious“ zu sehen ist, wie er in der XML-Datei spezifiziert ist. Zudem kann das Mapping auch für den Export einer ORDERRESPONSE genutzt werden, indem das Mapping in die andere Richtung angewendet wird. Dafür müssen die in der durch den Import erstellten Auftragsbestätigung enthaltenen ingenious-Bezeichner und -Werte wieder in die andere Richtung zurückübersetzt und in eine XML-Datei geschrieben werden.

Im Use Case „Varianten hinzufügen“, der den Use Case „Stücklisten-Artikel konfigurieren“ erweitert, können zudem eine oder mehrere Varianten angegeben werden. Hat der Nutzer zum Beispiel einen Stücklisten Artikel „A“, wovon die Varianten „A1“ und „A2“ existieren, die Konfiguration aber nur für Variante „A1“ gelten soll, dann kann der Nutzer das durch diesen Use Case festlegen.

Die beiden Use Cases „Einstellungen für ORDER vornehmen“ und „Einstellungen für ORDERRESPONSE vornehmen“ dienen dem Nutzer zur genauen Festlegung, was mit den Daten aus der Auftrags-XML-Datei geschieht, beziehungsweise wie die XML-Datei der Auftragsbestätigung aussehen soll. Dazu kann der Nutzer für jeden Knoten des jeweiligen Teilbereichs individuelle Einstellungen vornehmen.

Der Use Case „Auftrags-Import durchführen” wird entweder manuell vom Nutzer durch das Ausführen eines Skripts ausgelöst oder automatisch durch den Scheduler. Sollte der Import für der im oben beschriebenen Ordner liegenden Quelldateien erfolgreich abgelaufen sein, ist das Ergebnis dieses Use Cases eine oder mehrere Auftragsbestätigungen, welche der Nutzer in der Detailansicht der Projekte in „ingenious“ einsehen kann.

Der Use Case „Auftragsbestätigungs-Export durchführen“ wird in der eben beschriebenen Detailansicht der Projekte ausgelöst. Aus der angezeigten Auftragsbestätigung wird mit den Einstellungen, die durch den Use Case „Einstellungen für ORDERRESPONSE vornehmen“ beschrieben wurden, eine XML-Datei erstellt, die dem openTRANS®-Teilbereich ORDERRESPONSE entspricht.

3.2.2.2 Aktivitätsdiagramme

Der Use Case „Import durchführen“ zeigt den eigentlichen Prozess des Imports nur aus Anwendersicht. Sobald alle Einstellungen getroffen wurden, also alle anderen Use Cases stattgefunden haben, läuft der Import aus Sicht eines Nutzers automatisch ab. Daraus allein lassen sich also keine Schlüsse für die benötigte Systemarchitektur ziehen. Um diese Lücke zu schließen, eignet sich die Erstellung eines Aktivitätsdiagramms.
Ein Aktivitätsdiagramm ist eine Möglichkeit zur Modellierung von Aktionen in einem System. Hierbei werden verschiedene UML-Darstellungselemente genutzt, um den Ablauf eines Vorgangs zu beschreiben (Balzert 2009, S. 236f.).

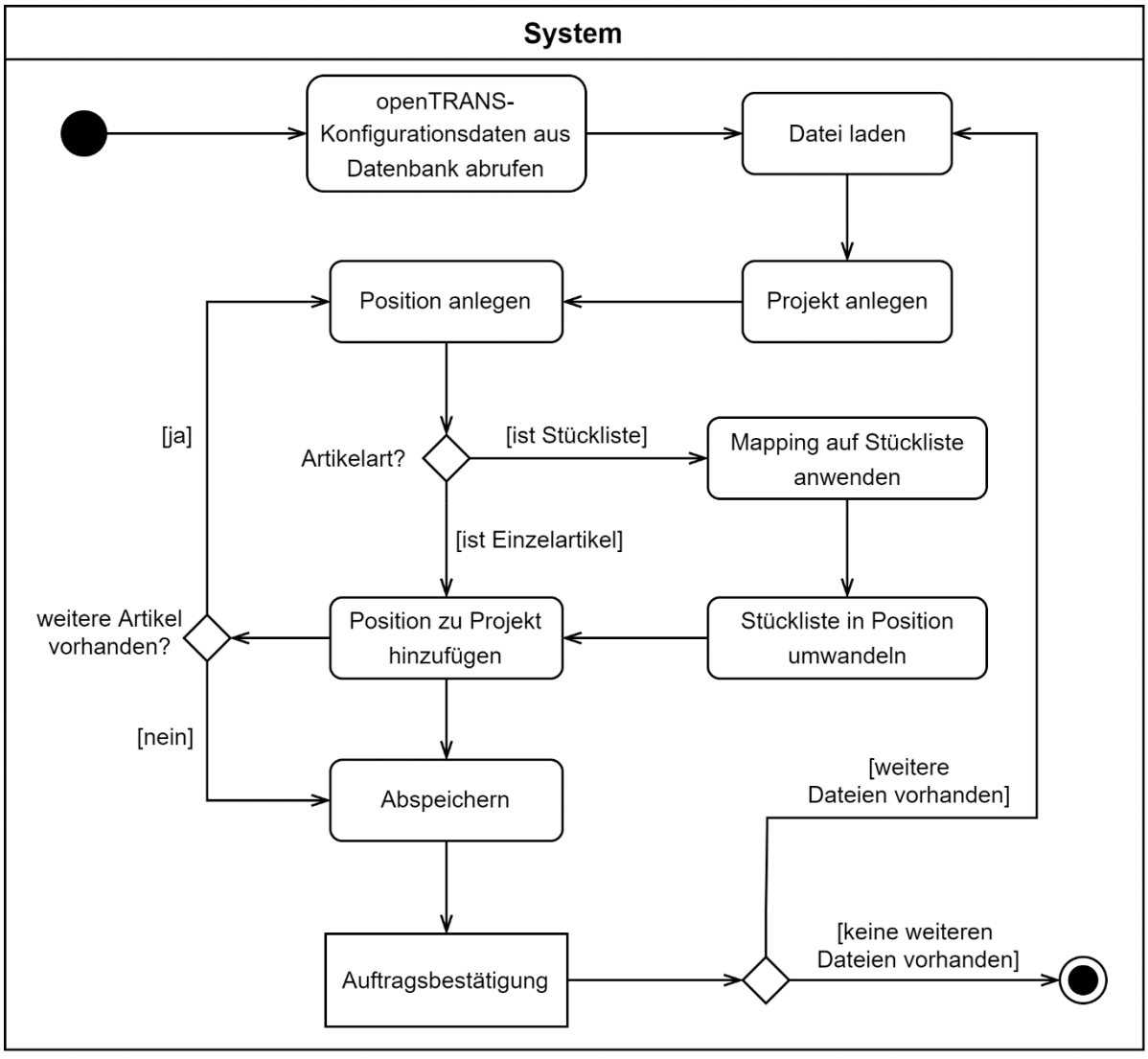


Abbildung 7: Aktivitätsdiagramm für den Ablauf des Imports
Quelle: Eigene Darstellung

Abbildung 7 zeigt das Aktivitätsdiagramm für den Use Case „Import durchführen“. Als Vorbereitung für den Import muss bestimmt werden, für welche Kontakte der Import durchgeführt werden soll. Dazu werden die Konfigurationsdaten, die in den restlichen Use Cases erstellt wurden, aus der Datenbank geladen. Für jeden Kontakt, für den Konfigurationsdaten existieren, kann nun der Importprozess gestartet werden.

Dafür werden zunächst alle Dateien aus dem Importverzeichnis geholt. Für jede Datei werden alle darin enthaltenen Informationen gemäß den getätigten Einstellungen für die openTRANS®-Elemente (siehe Pflichtenheft F40 und F50) in die Software überführt. Danach wird ein Projekt pro Datei angelegt, in dem Fall eine Auftragsbestätigung. Für das Füllen der Auftragsbestätigung mit Positionen muss zunächst für jede aus der Datei eingelesene Position geprüft werden, ob es sich um einen Einzelartikel oder eine Stückliste handelt. Ist es ein Einzelartikel, so kann dieser direkt als Position zur Auftragsbestätigung hinzugefügt werden. Handelt es sich um eine Stückliste, muss zunächst noch das Mapping (siehe Pflichtenheft F30 – F34) darauf angewendet werden. Erst danach kann die Position hinzugefügt werden.

Das Ergebnis nach Hinzufügen aller Positionen und Abspeichern des Projektes in der Datenbank ist eine Auftragsbestätigung, welche in „ingenious“ betrachtet und auf Korrektheit überprüft werden kann. In den Anlagen befindet sich ein Beispiel einer durch diesen Prozess erstellten Auftragsbestätigung.

3.3 Grafische Benutzeroberfläche

Die Anfertigung einer grafischen Benutzeroberfläche (GUI) bietet zusammen mit den Ergebnissen der Anforderungsanalyse eine umfassende Grundlage für die Entwicklung einer Softwarearchitektur. Die GUI für diese Arbeit wurde, wie der Rest von „ingenious“ mit Windows Forms erstellt. Der in die Entwicklungsumgebung Visual Studio integrierte Designer ermöglicht ein einfaches Zusammenstellen einer Oberfläche mittels Drag-and-drop einzelner GUI-Komponenten.

Die in Hinblick auf die Anforderungen erstellte Benutzeroberfläche lässt sich grundlegend in drei Teile gliedern:

- Grundeinstellungen (Abbildung 8)
- Artikel-Liste (Abbildung 9)
- openTRANS®-Einstellungen (Abbildung 12)

3.3.1 Grundeinstellungen

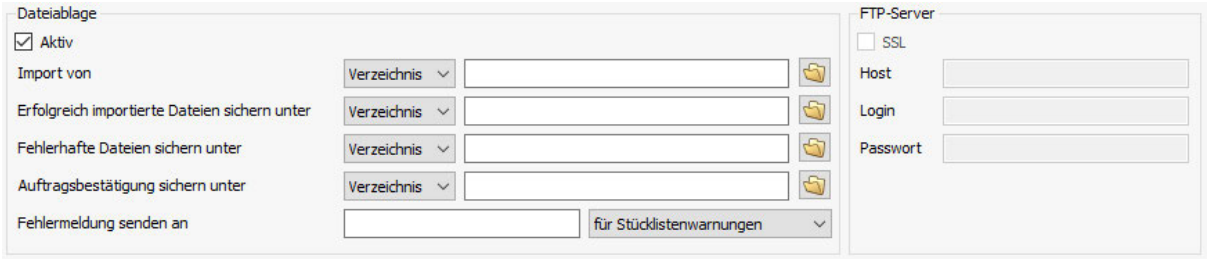


Abbildung 8: Grundeinstellungen
Quelle: Eigene Darstellung

Abbildung 8 zeigt die Grundeinstellungen für den Import. Sie sollen die Anforderungen F21 und F22 erfüllen. Die Checkbox „Aktiv“ zeigt an, ob der Import für diesen Kontakt durchgeführt werden soll oder nicht. Die restlichen Einstellungsmöglichkeiten sind eine Umsetzung des Use-Cases „Importeinstellungen vornehmen“, weshalb ihre Bedeutungen an dieser Stelle nicht noch einmal betrachtet werden. Sollte links im Teil „Dateiablage“ nicht die Option „FTP-Server“ gewählt sein, so wie in Abbildung 8 zu sehen, dann wird der Bereich für die FTP-Einstellungen ausgegraut.

3.3.2 Artikel-Konfigurationen-Liste

Mit diesem Bereich der GUI sollen die Anforderungen F30 bis F36 erfüllt werden. Hier werden primär Konfigurationen für Stücklisten-Artikel angezeigt.

Abbildung 9: Artikel-Konfigurationen-Liste
Quelle: Eigene Darstellung

Die Buttons auf der rechten Seite in Abbildung 9 bieten (von oben nach unten) Möglichkeiten zur Erstellung, Löschung, Duplizierung und Bearbeitung der Konfiguration. Ein Klick auf Button 1 „Erstellung“ und Button 4 „Bearbeitung der Konfiguration“ öffnet ein weiteres Fenster, welches in Abbildung 10 gezeigt wird.

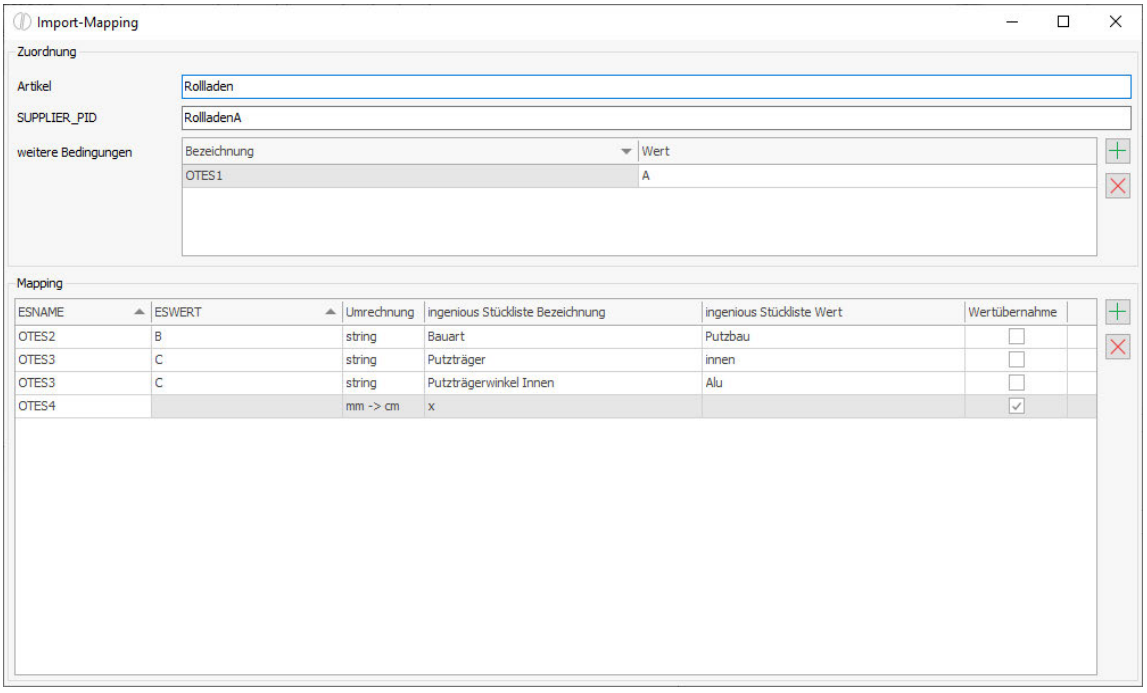


Abbildung 10: Mapping-Ansicht
Quelle: Eigene Darstellung

Dieses Fenster ist eine Umsetzung der durch die Use Cases „Stücklisten-Artikel konfigurieren“, „Varianten hinzufügen“ und „Mapping erstellen“ beschriebenen Einstellungsmöglichkeiten.

Zur Vereinfachung des Hinzufügens von Mapping-Zeilen zu der im unteren Teil von Abbildung 10 befindlichen Tabelle, kann mithilfe eines Klicks auf die letzte Spalte ein weiteres, hier in Abbildung 11 gezeigtes Fenster geöffnet werden.

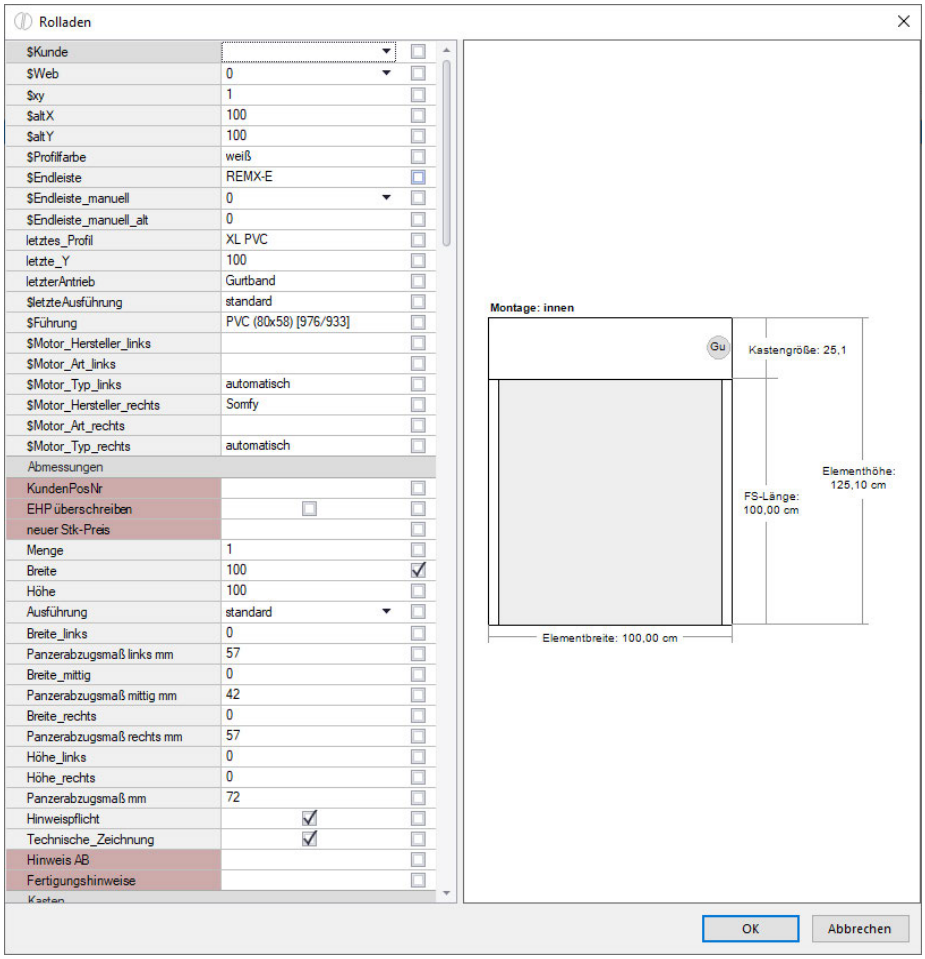


Abbildung 11: Stücklisten-Konfigurationsansicht
Quelle: Eigene Darstellung

Dieses Fenster ist bereits Bestandteil von ingenious und wurde lediglich im Rahmen dieser Arbeit wiederverwendet, um die im Folgenden beschriebene Funktionalität umsetzen zu können.

Pro gewählter Option wird nach dem Klicken auf „OK“ eine Zeile mit dem entsprechenden ingenious-Bezeichner, in Abbildung 11 beispielhaft die Breite und der Wert (hier 100) zur Tabelle hinzugefügt. Dies ermöglicht auch das Zuweisen eines externen Bezeichners zu

mehreren ingenious-Stücklisten-Bezeichnern mit nur einem Klick, indem in der Stücklisten-Konfigurationsansicht mehrere Elemente ausgewählt werden.

3.3.3 openTRANS®-Einstellungen

In diesem Abschnitt wird die Umsetzung der beiden Use Cases „Einstellungen für ORDER vornehmen“ und „Einstellungen für ORDERRESPONSE vornehmen“ gezeigt.
In den openTRANS®-Einstellungen können feinere Einstellungen zur Konfiguration des Imports getätigt werden. Für jedes openTRANS®-Element kann gewählt werden, ob und wohin dieses beim Import gespeichert werden soll. Damit sollen die Anforderungen F40 und F50 erfüllt werden.

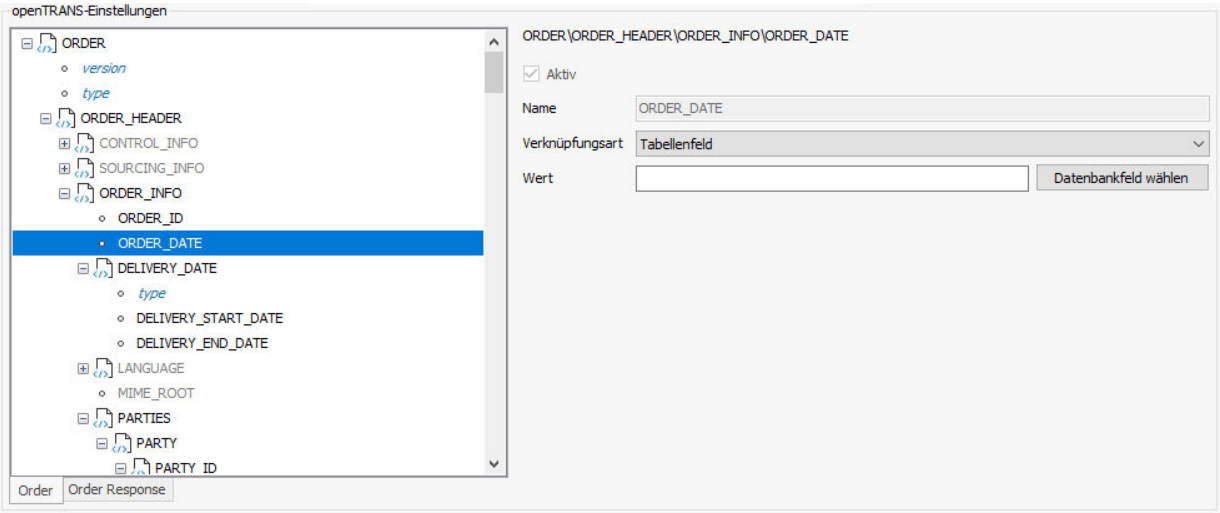


Abbildung 12: openTRANS®-Einstellungen
Quelle: Eigene Darstellung

In Abbildung 12 sind die Einstellungen für openTRANS® zu sehen. Links in der Abbildung ist ein Baum dargestellt, der alle Elemente des Teilbereichs ORDER des openTRANS®-Standards enthält. Mit den Reitern unten links lässt sich zwischen dem ORDER-Baum und dem ORDERRESPONSE-Baum umschalten. Wird ein Element gewählt, dann werden - wie in Abbildung 12 rechts zu sehen - Informationen und Einstellungsmöglichkeiten dazu angezeigt. Attribute werden zur Unterscheidung von Kindelementen mit blauer Schrift angezeigt. Elemente und Attribute werden aufgrund ihrer Darstellung in einem Baum im Folgenden zusammenfassend auch als Knoten bezeichnet.

Bedienelement	Option	ORDER	ORDERRESPONSE
Checkbox „Aktiv“	aktiviert	Knoten aus XML-Datei wird importiert	Knoten wird in XML-Datei eingefügt
Checkbox „Aktiv“	deaktiviert	Knoten aus XML-Datei wird nicht importiert	Knoten wird nicht in XML-Datei eingefügt
Verknüpfungsart	Tabellenfeld	Wert des Knotens wird in Datenbankfeld gespeichert	Wert des Knotens in XML-Datei kommt aus Datenbankfeld
Verknüpfungsart	Benutzerfeld (spezielle Form eines Datenbankfelds)	Wert des Knotens wird in Benutzerfeld gespeichert	Wert des Knotens in XML-Datei kommt aus Benutzerfeld
Verknüpfungsart	Fester Wert	bei ORDER nicht vorhanden	Wert des Knotens in XML-Datei wird aus Textbox übernommen
Verknüpfungsart	Skript	bei ORDER nicht vorhanden	Wert des Knotens in XML-Datei kommt aus einem Skript
Verknüpfungsart	wie bei ORDER	bei ORDER nicht vorhanden	gleiche Verknüpfungsart, wie bei ORDER angegeben wurde

Tabelle 6: Einstellungsmöglichkeiten für Knoten

Tabelle 6 erklärt die Bedienelemente aus Abbildung 12, welche durch den Nutzer geändert werden können. Grundsätzlich sind diese bei ORDER und ORDERRESPONNSE identisch. Da sich aber die Bedeutungen unterscheiden, wird in Tabelle 6 danach differenziert. Zusätzlich zum Inhalt der Tabelle sei noch erwähnt, dass für Muss-Elemente die Checkbox „Aktiv“ ausgegraut ist und immer aktiv bleibt (siehe Abbildung 12), weil diese - wie bereits in Kapitel 2.3 erklärt - laut openTRANS-Spezifikation immer in der XML-Datei enthalten sein müssen.

Beim ORDER-Baum sind für das Element PARTY_ROLE andere Einstellungsmöglichkeiten als bei den anderen Knoten vorhanden, weil „ingenious“ mehrere Adresstypen unterscheidet und diese gemäß verschiedener entsprechender PARTY_ROLEs beim Import berücksichtigt werden müssen.

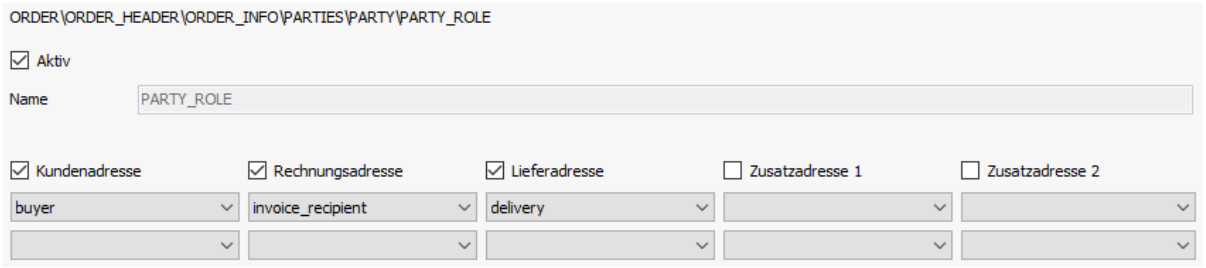


Abbildung 13: Einstellungsmöglichkeiten Spezialfall PARTY_ROLE
Quelle: Eigene Darstellung

Abbildung 13 zeigt die für PARTY_ROLE abweichenden Optionen. Das openTRANS-Element PARTY beinhaltet Informationen zu einem Geschäftspartner. Einer der Kindknoten, PARTY_ROLE, beschreibt die Rolle des Geschäftspartners bei der Transaktion. Dafür gibt es eine Liste von Rollen, die als Wert gültig sind, beispielsweise „buyer“ für ein einkaufendes Unternehmen (Schmitz et al. 2009, S. 50).

In „ingenious“ existieren fünf Adresstypen, welche im unteren Bereich von Abbildung 13 in Form von Checkboxen zu sehen sind. Pro Adresstyp hat der Nutzer die Möglichkeit, bis zu zwei Rollen aus der openTRANS-Spezifikation in den unter den Checkboxen liegenden Drop-down-Feldern zu wählen. Ist eine Checkbox aktiv und in den dazugehörigen Checkboxen mindestens ein Wert gewählt, wird beim Import eine Adresse mit dem entsprechenden Adresstyp aus „ingenious“ erstellt und mit den Werten aus dem PARTY-Element der zu importierenden Datei gefüllt, sofern dort eine PARTY_ROLE mit dem Wert aus einer der Checkboxen existiert. Nach den Einstellungen aus Abbildung 13 würde beispielsweise die Lieferadresse mit den Adressdaten aus der PARTY gefüllt, die die PARTY_ROLE „delivery“ besitzt.

Beim Klicken auf „Datenbankfeld wählen“ öffnet sich ein weiteres Fenster, welches in Abbildung 14 zu sehen ist.

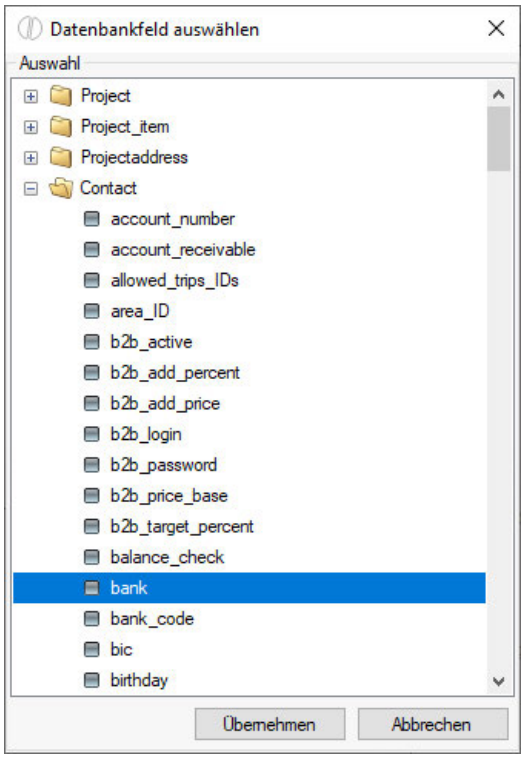


Abbildung 14: Fenster Datenbankfeld wählen
Quelle: Eigene Darstellung

Hier kann der Nutzer einfach ein Feld aus einer der Tabellen in der Datenbank wählen, in welches beim Import der Wert des ausgewählten Elements gespeichert werden soll. Beim Klick auf „Übernehmen“ wird das gewählte Feld in die Textbox „Wert“ (siehe Abbildung 12) eingetragen.
Der Baum aus Abbildung 12 kann zudem mit den in Abbildung 15 zu sehenden Buttons bearbeitet werden.

Abbildung 15: Buttons für Bearbeiten des openTRANS®-Einstellungen-Baums
Quelle: Eigene Darstellung

Sofern es durch den openTRANS®-Standard erlaubt ist, ein Element also mehrfach vorkommen darf, kann mit dem Button „Knoten hinzufügen“ eine Kopie des aktuell

ausgewählten Elements in den Baum eingefügt werden. Elemente können genau so, sofern mindestens zwei Elemente mit diesem Namen vorhanden sind, wieder entfernt werden. Die restlichen beiden Buttons erfüllen eine ähnliche Funktion, nur dass sie für einen Spezialfall des openTRANS®-Standards gedacht sind: dem Element PRODUCT_COMPONENTS. Dieses besitzt das Kindelement PRODUCT_COMPONENT, welches wiederum das Element PRODUCT_COMPONENTS enthält. (Schmitz et al. 2009, S. 181). Dadurch ergibt sich eine Art unendliche Rekursion, welche im Code gesondert behandelt werden muss.

Alle gezeigten Teile der GUI befinden sich in der Detailansicht eines Kontakts. Diese wird angezeigt, wenn der Nutzer auf einen in ingenious gespeichert Kontakt klickt. Links in der Navigationsleiste ist die neue Option „openTrans Import“ zu sehen. Beim Klick auf diese wird das Fenster, was in Abbildung 16 dargestellt ist, angezeigt. Hier sind alle bisher in diesem Kapitel beschriebenen GUI-Bestandteile in ihrer endgültigen Anordnung zu sehen.

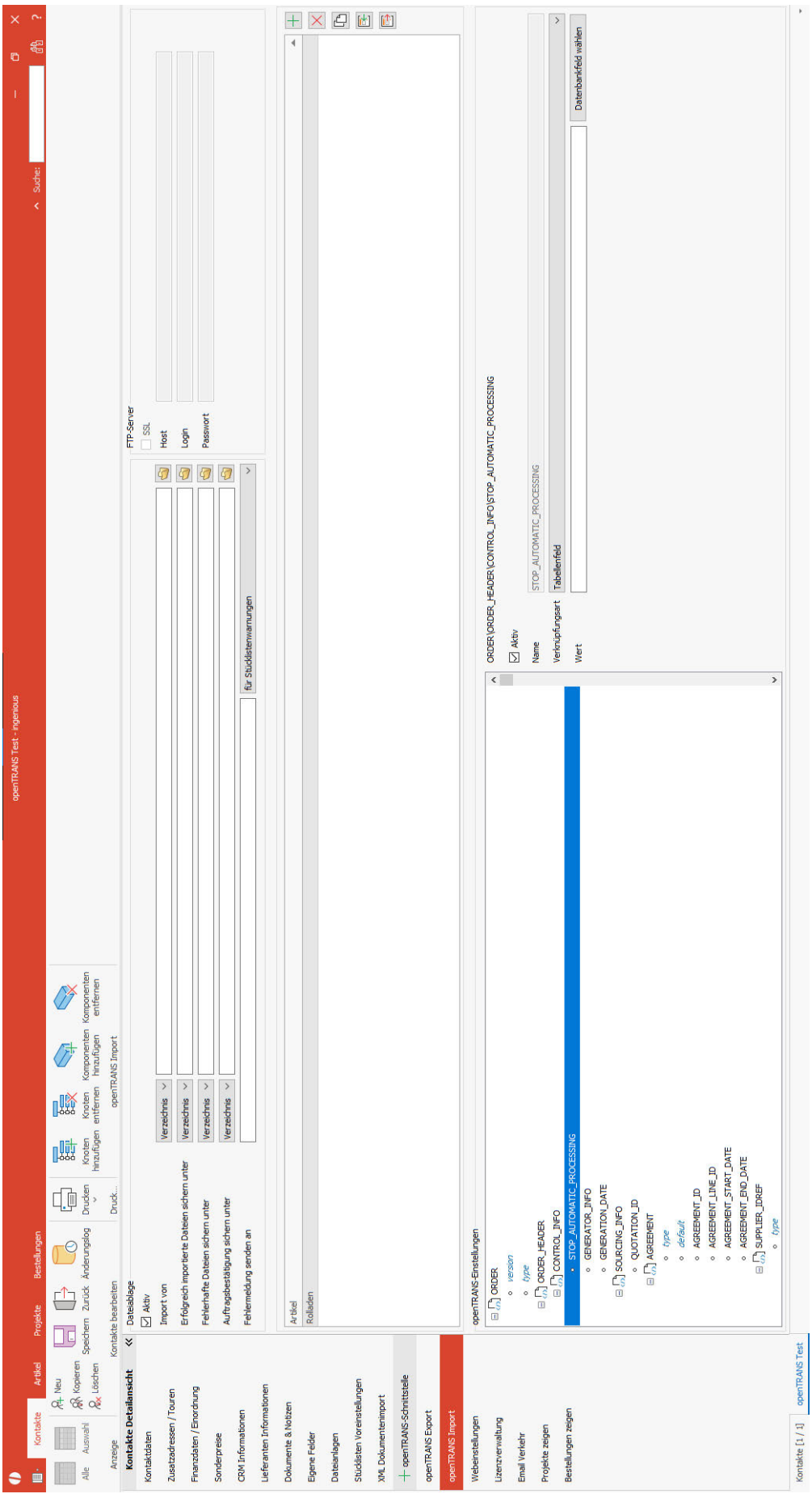


Abbildung 16: Gesamte GUI
Quelle: Eigene Darstellung

3.3.4 Auftragsbestätigungs-Export

Zusätzlich zu der GUI in der Kontakte-Detailansicht wurde für den Auftragsbestätigungs-Export in der Projekte-Detailansicht eine Option zum Durchführen des Exports ergänzt. In einer Sidebar werden alle relevanten Informationen zu einer Auftragsbestätigung angezeigt.

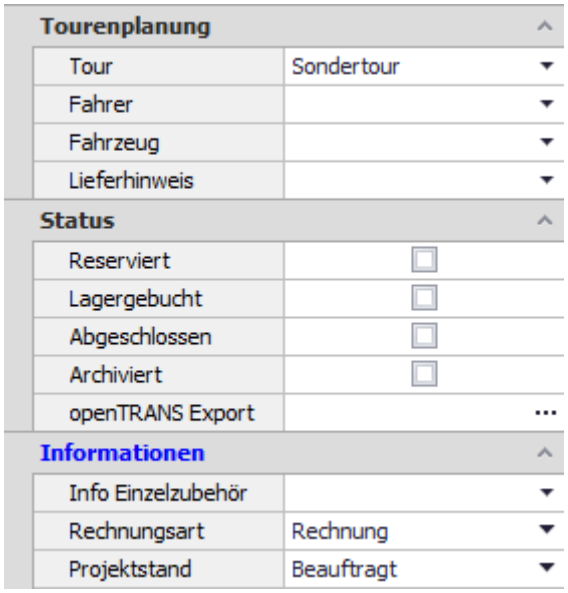


Abbildung 17: Ausschnitt der Sidebar in der Projekte-Detailansicht
Quelle: Eigene Darstellung

Abbildung 17 zeigt einen Ausschnitt aus dieser Sidebar. Unter „Status“ befindet sich die neu erstellte Option für den Auftragsbestätigungs-Export. Werden die 3 Punkte rechts im Feld geklickt, dann wird der Exportprozess gestartet und das aktuelle Datum in das Feld eingetragen. Ist einmal ein Datum eingetragen worden, dann bedeutet das, dass die aktuell angezeigte Auftragsbestätigung bereits exportiert wurde. Ein erneuter Klick auf die 3 Punkte ist dann wirkungslos, um zu verhindern, dass die gleiche Auftragsbestätigung mehrmals exportiert wird.

3.4 Softwarearchitektur

3.4.1 Datenbankstruktur

Für die Repräsentation der Daten in der Datenbank sind zwei neue Tabellen vorgesehen. In der ersten werden die Daten der in Kapitel 3.3 beschriebenen GUI-Bereiche Grundeinstellungen und openTRANS®-Einstellungen gespeichert. Die Artikel-Liste wird durch die zweite Tabelle in der Datenbank abgebildet.

Feldname	**Datentyp**	**Beschreibung**
ID	uniqueidentifier (GUID)	Identifikator des Datenbankeintrags
contact_ID	uniqueidentifier (GUID)	ID des Kontakts
order_active	bit (bool)	Status Checkbox „Aktiv“
order_config	varbinary (byte[])	Grundeinstellungen
order_tree	varbinary (byte[])	openTRANS®-Einstellungen Teilbereich ORDER
orderresponse_tree	varbinary (byte[])	openTRANS®-Einstellungen Teilbereich ORDERRESPONSE

Tabelle 7: Datenbanktabelle „OpenTrans_import“

Tabelle 7 „OpenTrans_import“ zeigt die erste der beiden neuen Datenbanktabellen. Hier wird für jeden Kontakt, für den ein openTRANS®-Import konfiguriert werden soll, ein Eintrag angelegt. Der Datentyp ist als SQL-Datentyp und in Klammern als C#-Datentyp angegeben. Der Primärschlüssel *ID* wird beim erstmaligen Anlegen des Datenbankeintrags neu erzeugt. Die *contact_ID* stammt aus einer weiteren, bereits vorhanden Datenbanktabelle und ist damit Fremdschlüssel. Mit ihr kann beim Öffnen eines Kontaktes eindeutig identifiziert werden, welcher Datensatz aus der Datenbank geladen werden muss. Da der Status der Checkbox „Aktiv“ ein Boolean ist, kann dieser direkt für *order_active* direkt übernommen werden. Für Grundeinstellungen und die beiden openTRANS®-Einstellungen wurde als Datentyp ein Byte-Array gewählt, da „ingenious“ bereits eine Implementierung besitzt, die Klassen, welche solche Daten repräsentieren, effizient speichert.

Feldname	Datentyp	Beschreibung
ID	uniqueidentifier (GUID)	Identifikator des Datenbankeintrags
contact_ID	uniqueidentifier (GUID)	ID des Kontakts
item_ID	uniqueidentifier (GUID)	ID des Artikels
identifier	varbinary (byte[])	Nähere Beschreibung des Artikels (Abbildung 10 oben, Abschnitt „Zuordnung“)
configuration	varbinary (byte[])	Mapping-Tabelle (Abbildung 10 unten, Abschnitt „Mapping“)

Tabelle 8: Datenbanktabelle OpenTrans_import_mapping

Tabelle 8 „OpenTrans_import_mapping“ ist die zweite der neuen Datenbanktabellen. Hier wird für jeden Artikel, der zur Artikel-Liste hinzugefügt wird, ein Eintrag angelegt. Die Felder *ID* und *contact_ID* haben die gleiche Bedeutung wie bei der oben beschriebenen Tabelle „OpenTrans_import“. Mit dem Fremdschlüssel *item_ID* wird eindeutig ein Artikel aus der bereits vorhandenen Tabelle „Item“ referenziert. Damit kann sichergestellt werden, dass im *identifier* nur Artikel gespeichert werden, die auch wirklich in der Datenbank existieren. Für Identifier und configuration wurde aus oben genannten Gründen wieder das Byte-Array als Datentyp gewählt.

3.4.2 Klassendiagramme

Nachdem nun sowohl die Anforderungen, die GUI und die Datenbankstruktur spezifiziert wurden, können daraus Klassen abgeleitet werden. Diese werden in Form von zwei sich ergänzenden Klassendiagrammen gezeigt.

3.4.2.1 Klassendiagramm Grafische Benutzeroberfläche

Im Folgenden ist zunächst das Klassendiagramm dargestellt, welches die für die GUI benötigten Klassen enthält.

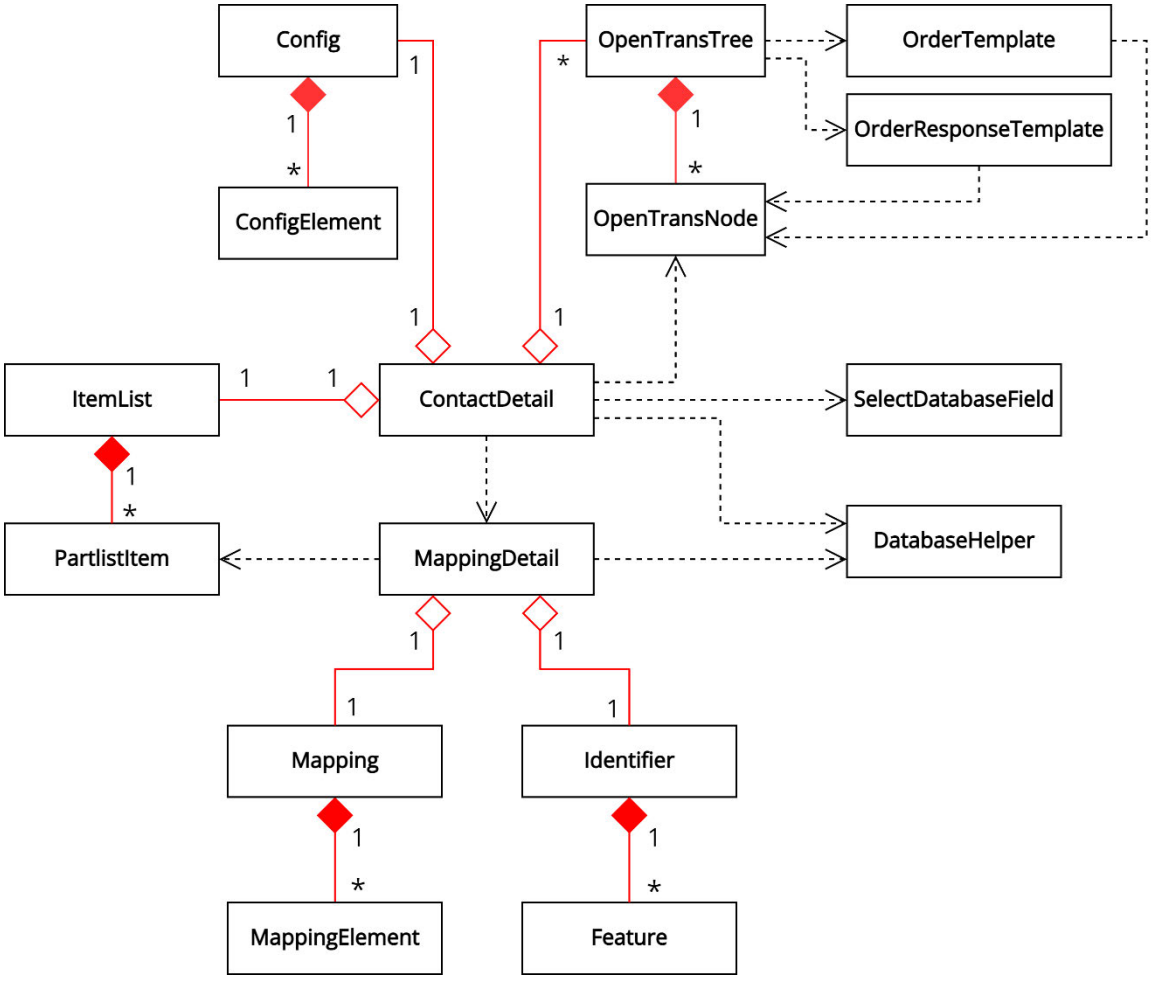


Abbildung 18: Klassendiagramm der GUI
Quelle: Eigene Darstellung

Abbildung 18 zeigt das Klassendiagramm der GUI. Zur Wahrung der Übersichtlichkeit enthält es nur die Namen der Klassen; Attribute und Methoden wurden weggelassen.

Das zentrale Element, welches die GUI verwaltet, ist die partielle Klasse ContactDetail. Partiell bedeutet, dass der Code der Klasse auf mehrere Dateien aufgeteilt werden kann. (Microsoft 2022b). ContactDetail existiert bei Ingenious bereits und ist dort für die gesamte Detailansicht der Kontakte verantwortlich. Das heißt, dass das gesamte Fenster, welches in Abbildung 16 dargestellt ist, durch diese Klasse implementiert ist. Demzufolge wurde ContactDetail um eine weitere Quellcode-Datei erweitert, die GUI für den openTRANS®-Import steuert.

Viele der Klassen wurden nach dem Entwurfsmuster „Liste“ erstellt. Dieses ist nützlich, wenn ein Objekt modelliert werden muss, welches aus gleichartigen Einzelteilen besteht. Die Einzelteile sind dem Objekt fest zugeordnet, können aber auch wieder entfernt werden. Das ermöglicht eine flexible Verwaltung von vielen Objekten unter einem Aggregat-Objekt (Balzert 2009, S. 551). Folgende Klassen nutzen dieses Entwurfsmuster:

- Config und ConfigElement
- ItemList und PartlistItem
- Mapping und MappingElement

Für die Abbildung der Grundeinstellungen wurde die Klasse Config erstellt. Sie enthält eine Liste von ConfigElement-Objekten. Jedes ConfigElement speichert den Namen eines GUI-Elements in den Grundeinstellungen und den darin eingetragenen Wert. So ist zum Beispiel die Dropdown-Liste rechts neben dem Text „Import von“ (siehe Abbildung 8) ein ConfigElement, genauso wie das Textfeld daneben.

Über die Klasse ItemList kann auf eine Liste von PartlistItem-Objekten zugegriffen werden. Ein PartlistItem besitzt eine Record-ID, eine Artikel-ID und einen Matchcode. Die Record-ID entspricht dem Feld ID und die Artikel-ID dem Feld item_ID der Tabelle OpenTrans_import_mapping (siehe Tabelle 8). Der Matchcode in Form des Namens des Artikels stammt aus der in Kapitel 3.4.1 erwähnten Item-Tabelle. Dieser wird in der Artikel-Liste aus Abbildung 9 angezeigt und dient dem Nutzer zur einfachen Erkennung, um welchen Artikel es sich handelt.
Für die Repräsentation einer Zeile aus der Mapping-Tabelle aus Abbildung 10 wurde die Klasse MappingElement entworfen. Für jede der Spalten der Tabelle existiert eine

entsprechende Eigenschaft im MappingElement. Die Gesamtheit aller Mapping-Elemente eines Stücklisten-Artikels wird in der Klasse Mapping aggregiert.

Die Klasse MappingDetail ist für die Darstellung des Fensters aus Abbildung 10 verantwortlich und damit auch für die Verwaltung des Identifiers, der MappingElemente und die Erstellung beziehungsweise das Bearbeiten eines PartlistItems. Die Fenster in „ingenious“ sind asynchron implementiert, das heißt, wenn aus einem Fenster heraus ein zweites geöffnet wird, wartet das erste darauf, dass das zweite geschlossen wird. Weil dadurch das erste Fenster nicht auf Eigenschaften oder Methoden des zweiten Fensters zugreifen kann, ist darüber ein einfacher Austausch von Daten zwischen den Fenstern nicht möglich. Zum Zurückgeben des in MappingDetail erstellten oder bearbeiteten PartlistItems an ContactDetail wird daher ein Delegat genutzt. Ein Delegat gibt Methoden als Parameter an andere Methoden weiter (Microsoft 2022a). MappingDetail besitzt ein Delegat, welchem eine Methode aus ContactDetail übergeben wird. Beim Klicken auf „OK“ im Fenster wird der Delegat ausgeführt und damit auch die Methode in ContactDetail. Dieser wird das PartlistItem übergeben, welches in dem Fenster konfiguriert wurde. Das PartlistItem kann somit zur weiteren Verarbeitung genutzt werden, zum Beispiel zum Hinzufügen zur Artikel-Liste (siehe Abbildung 9).

Die Klasse OpenTransNode repräsentiert einen Knoten, so wie er in der openTRANS®-Spezifikation von Schmitz et al. 2009 beschrieben wird. So wird beispielsweise die Muss-Eigenschaft (siehe Kapitel 2.3) durch einen Boolean dargestellt oder die Kardinalität durch einen Integer. Außerdem existieren Verweise auf den Elternknoten und die Kindknoten. Mehr zum Aufbau einer OpenTransNode befindet sich in Kapitel 4.

Die Knoten ergeben zusammengefasst einen Baum. Dieser wird durch die Klasse OpenTransTree abgebildet. Sie bildet das Bindeglied zwischen der openTRANS®-Spezifikation, dem Importprozess und der GUI, welche mit den Windows-Forms-Klassen TreeView und TreeNode, wie in Abbildung 12 auf der linken Seite zu sehen, angezeigt werden. Da ein OpenTransNode bereits die Information enthält, welche Kindknoten zu ihm gehören, ist es ausreichend, in der Klasse OpenTransTree als einzigen Knoten einen OpenTransNode zu speichern, der die Wurzel des Baumes darstellt. Mittels Rekursion ist es möglich, den

ganzen OpenTransTree ausgehend vom Wurzelknoten zu durchlaufen und währenddessen zum Beispiel den Baum in der GUI zu erstellen oder die zu importierenden Dateien einzulesen.
Um die im openTRANS®-Standard beschriebenen Bäume mit all ihren Knoten als einen OpenTransTree abzubilden, wurden die Klassen OrderTemplate und OrderResponseTemplate erstellt. In diesen werden alle Elemente im jeweiligen Teilbereich des Standards mit ihren dazugehörigen Eigenschaften als OpenTransNodes initialisiert und anschließend als Baum zusammengefügt. Dadurch wird eine Art Vorlage geschaffen; eine Baumstruktur, die keine weiteren Daten enthält, die allerdings genutzt werden kann, um einen OpenTransTree zu erstellen. Dazu muss lediglich der Wurzelknoten des OpenTransTrees mit dem Wurzelknoten der Vorlage initialisiert werden. Diese beiden Klassen werden genutzt, um beim ersten Aufrufen der neu erstellten Seite (siehe Abbildung *16*) einen neuen Baum zu generieren.

Die Klasse SelectDatabaseField implementiert das Fenster aus Abbildung 14. Hier wird, wie in MappingDetail, ein Delegat eingesetzt, um Informationen aus dem Fenster zurückzugeben. In diesem Fall ist es das ausgewählte Datenbankfeld. Die Ordner in Abbildung 14 sind Datenbanktabellen, die Unterpunkte alle zur Tabelle gehörigen Felder. Mit einem Klick auf OK wird das Fenster geschlossen und das gewählte Tabellenfeld in die Textbox „Wert“ (siehe Abbildung 12) eingetragen. Welche Tabellen mit im Fenster angezeigt werden sollen, ist durch eine variable Anzahl von Parametern wählbar. Dadurch kann diese Klasse nicht nur für diesen Fall angewendet werden, sondern auch potenziell in der ganzen Anwendung genutzt werden.

Die statische Klasse DatabaseHelper wurde entworfen, um Objekte in der Datenbank zu speichern und daraus wiederherzustellen. In der darin enthaltenen statischen Store()-Methode wird ein Objekt mittels in „ingenious“ bereits vorhandenen Klassen in ein Byte-Array umgewandelt, welches dann an anderer Stelle in ein Datenbankfeld gespeichert wird. Dadurch, dass die Klasse und die Methode statisch sind, kann sie an beliebiger Stelle mit jeglichen Objekten eingesetzt werden. Ein Aufruf sähe zum Beispiel so aus:
`DatabaseHelper.Store(config)`. Die Methode wird zur Speicherung von Objekten der Klassen Config, OpenTransTree, Identifier und Mapping genutzt. Als Gegenstück zu Store()

existiert die ebenfalls statische Methode Restore(). Sie nimmt ein Byte-Array und wandelt es in das ursprüngliche Objekt um. Die Funktionsweise ist ähnlich zur Store()-Methode, nur andersherum. Damit können Objekte oben genannter Klassen aus der Datenbank wiederhergestellt werden.

3.4.2.2 Klassendiagramm Import und Export

Für den eigentlichen Import der Aufträge und den damit zusammenhängenden Export von Auftragsbestätigungen wurde ein zweites Klassendiagramm erstellt (siehe Abbildung 19), um das erste für die GUI so übersichtlich wie möglich zu halten.

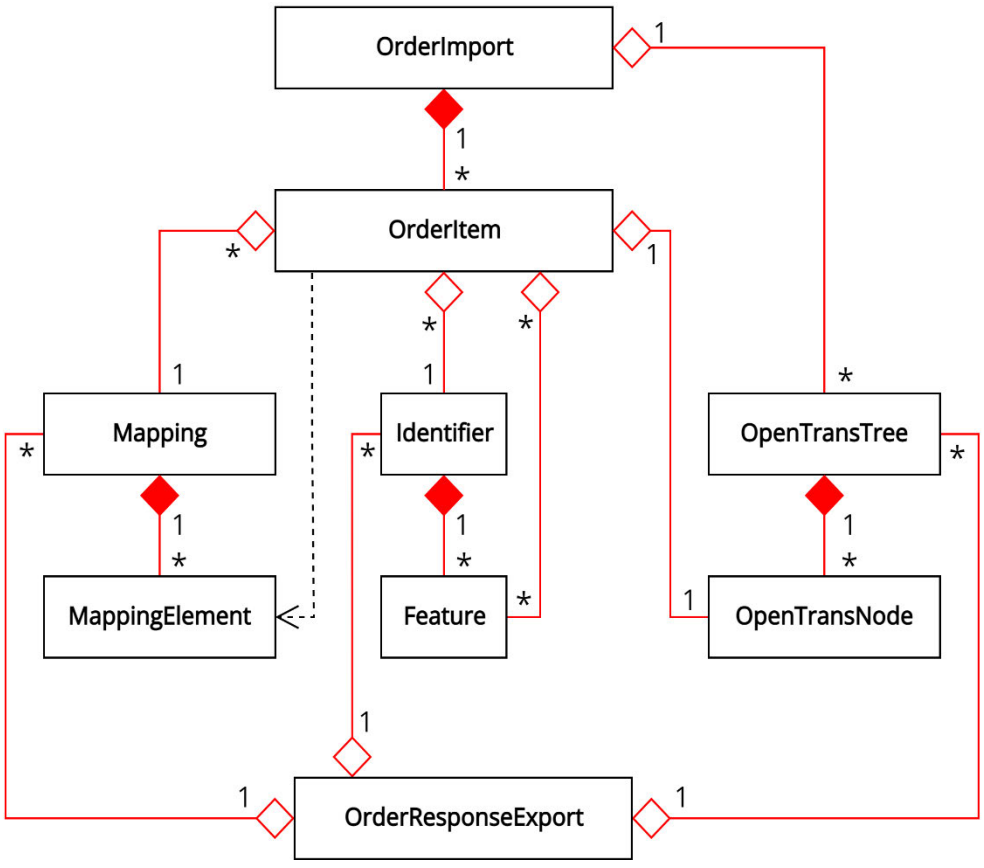


Abbildung 19: Klassendiagramm des Imports und des Exports
Quelle: Eigene Darstellung

Alle Klassen bis auf OrderImport, OrderResponseExport und OrderItem sind bereits aus dem vorherigen Abschnitt bekannt und werden daher an dieser Stelle nicht noch einmal betrachtet.

Die Klasse OrderImport steuert und führt den gesamten Importprozess durch. In verschiedenen Methoden werden die Schritte des Aktivitätsdiagramms aus Abbildung 7 abgearbeitet. Einer der Schritte ist es, die Artikel als Positionen zur Auftragsbestätigung hinzuzufügen. Um dies zu vereinfachen, werden Artikel mit der Klasse OrderItem modelliert. Die Klasse OrderItem enthält einen OpenTransNode, welcher beim Erstellen einer Instanz dieser Klasse mit einem ORDER_ITEM-Knoten initialisiert wird. In diesem sind durch das Einlesen des Auftrags alle Informationen zu einem Artikel aus der XML-Datei gespeichert. Unter Nutzung der gespeicherten Identifier kann der Artikel entweder als ein bestimmter Stücklistenartikel identifiziert werden oder als normaler Artikel. Ist es ein Stücklistenartikel, so kann darauf das gespeicherte Mapping angewendet werden. Das OrderItem wird anschließend in eine Position umgewandelt und zur Auftragsbestätigung hinzugefügt.

In der Klasse OrderResponseExport werden alle Informationen zu einer Auftragsbestätigung gesammelt, indem zunächst die im OpenTransTree der ORDERRESPONSE gespeicherten Einstellungen (siehe Kapitel 3.3.3) durchlaufen und die Werte der Knoten aus den angegebenen Quellen ausgelesen werden. Danach können die Positionen entsprechend des Mappings wieder in ORDER_ITEM-Knoten umgewandelt werden. Abschließend wird aus dem im eben beschriebenen Prozess resultierenden OpenTransTree eine XML-Datei erstellt.

4. Details zur Implementierung

In diesem Kapitel werden einige, ausgewählte Details der Implementierung anhand von Codebeispielen erklärt, die besondere Entwurfsentscheidungen oder das Erfüllen von Anforderungen darstellen.

4.1 Fluent Interface

Jede dem OpenTransNode zugewiesene Eigenschaft lässt sich mittels eines sogenannten Fluent Interface setzen. Der Begriff wurde erstmals von Martin Fowler verwendet. Kern dieser Programmiertechnik besteht aus der Umnutzung der üblichen Setter-Methode. Normalerweise wird in einer Setter-Methode nur ein Wert gesetzt, sie besitzt also keinen Rückgabewert. Mit dem Fluent Interface wird dies geändert. Es wird nicht nur ein Wert gesetzt, sondern auch ein Objekt zurückgegeben. Indem auf diesem Rückgabeobjekt wieder eine Methode ausgeführt wird, wird eine Verkettung von Operationen ermöglicht (Fowler 2005).
Ein Beispiel einer solchen Methode des Fluent Interfaces sieht folgendermaßen aus:

```
public OpenTransNode SetParent (OpenTransNode parent)
{
	Parent = parent;
	return this;
}
```

Listing 1: Beispiel einer Fluent-Interface-Methode

Listing 1 zeigt eine Fluent-Interface-Methode, so wie sie in der Klasse OpenTransNode implementiert wurde. Der Parent des OpenTransNodes, für den die Methode aufgerufen wurde, wird auf den übergebenen OpenTransNode gesetzt. Anschließend wird der Aufrufer wieder zurückgegeben. Die Erstellung eines OpenTransNodes mittels Verkettung solcher Fluent-Interface-Methoden könnte wie folgt aussehen:

```
OpenTransNode discountFactor = new OpenTransNode()
      .SetName("DISCOUNT_FACTOR")
      .SetParent(timeForPayment)
      .SetNodeType(NodeType.Project)
      .SetOpenTransDataType(OpenTransDataType.Float);
```

Listing 2: Erstellung eines OpenTransNodes mit Fluent Interface

Wie in Listing 2 zu sehen ist, kann ein komplexes Objekt, welches viele Eigenschaften besitzt, einfach durch die Verkettung verschiedener Methoden erstellt werden. Alle Eigenschaften werden zunächst im Konstruktor `OpenTransNode()` initialisiert. Anschließend können je nach Notwendigkeit beliebig viele Eigenschaften hinzugefügt werden. Im Gegensatz zu einem normalen Konstruktor ist die Reihenfolge der Angabe der Eigenschaften irrelevant. Auch ein Weglassen nicht benötigter Eigenschaften ist mit einem normalen Konstruktor nicht in diesem Ausmaß möglich. All dies macht das Fluent Interface zu einer flexiblen und leicht lesbaren Variante zur Erstellung von Objekten.

4.2 Skripte

Der Import kann entweder manuell oder automatisch gestartet werden. Beides funktioniert über Skripte.

Skripte bezeichnen in diesem Kontext ein in „ingenious“ enthaltenes Feature, welche die bestehenden Funktionen erweitern und größere Möglichkeiten der Individualisierung schaffen. Ein Skript ist ein in der Software individuell anpassbares Stück C#-Code, welches beim Speichern als ausführbare Assembly in die Datenbank abgelegt wird. Damit lässt sich, ohne neu kompilieren zu müssen und ohne die Software als Ganzes zu verändern, jederzeit C#-Code ausführen.

Das Skript, mit dem der Import manuell gestartet werden kann, sieht folgendermaßen aus:

```
public string OpenTransOrderImport()
{
	return OpenTransImport.OpenTransOrderImport.StartImport();
}
```

Listing 3: Skript für den manuellen Import

Wie in Listing 3 zu sehen ist, ist das Skript eine C#-Methode, die eine weitere Methode aufruft, dessen Inhalt allerdings für einen Nutzer von „ingenious“ nicht sichtbar ist. Das hat einerseits den Vorteil, dass sich die im Skript aufgerufene Methode in einer Entwicklungsumgebung debuggen lässt, andererseits erhält der Nutzer dadurch nur den Rückgabewert der Methode, was in diesem Fall das wünschenswerte Verhalten ist. Dieser Rückgabewert gibt dem Nutzer Feedback über den Import. Darin enthaltene Informationen sind beispielsweise eine Angabe, wie viele Aufträge importiert wurden oder im Falle eines nicht erfolgreichen Imports die Ursache des Fehlschlags.

Das Skript für den automatischen Import sieht dem des manuellen sehr ähnlich:

```
public void OpenTransOrderImport(out bool bError, out string sResult,
                                 out string sDescription)
{
	bError = false;
	sResult = "";
	sDescription = "";

	OpenTransOrderImport.StartImport(out bError, out sResult, out sDescription);
}
```

Listing 4: Skript für den automatischen Import

Der Unterschied des in Listing 4 gezeigten Skripts zum anderen Skript, ist, dass hier in der Importmethode einige weitere Rückgabewerte gesetzt werden. Grund für die Parameter ist die Verwendung des Skripts im Scheduler. Mit diesem lassen sich Skripte, die void als

Rückgabetyp und genau diese Parameter besitzen, automatisch und zeitgesteuert aufrufen. Dadurch ist auch Anforderung F80 erfüllt.

Das letzte neu eingeführte Skript ermöglicht die Veränderung von Daten in der vom Import erstellten Auftragsbestätigung.

```
public void OpenTransOrderImportBeforeSave(DataSets.DataSet_Projects dataset)
{
}
```

Listing 5: Skript für das Anpassen der Auftragsbestätigung

Listing 5 zeigt dieses Skript. Es wird immer vor dem Speichern der Auftragsbestätigung in der Datenbank aufgerufen. Der Methodenrumpf ist standardmäßig leer, da der Aufbau einer Auftragsbestätigung in „ingenious“ individuell angepasst werden und demzufolge kein allgemeingültiges Skript erstellt werden kann.

5. Zusammenfassung und Ausblick

5.1 Ergebnisse

Die im Rahmen dieser Arbeit erarbeiteten Anforderungen wurden vollständig erfüllt. Es wurde eine funktionsfähige Erweiterung für die Software „ingenious“ erstellt, die ein Importieren von Aufträgen in Form von XML-Dateien, die dem openTRANS®-Standard entsprechen, ermöglicht. Die importierten Aufträge sind dann als Auftragsbestätigungen einsehbar. Nach Überprüfung auf Korrektheit können diese als XML-Dateien exportiert werden, die dem openTRANS®-Teilbereich ORDERRESPONSE entsprechen.

Aus der Entwicklung der im Rahmen dieser Arbeit erschaffenen Schnittstelle lässt sich eine elementare Erkenntnis ziehen: es wichtig, eine flexible Softwarearchitektur zu erstellen. Diese sollte so aufgebaut sein, dass damit eine Brücke zwischen dem gegebenen Datenformat, in diesem Fall XML-Dateien, und der Darstellung auf Anwenderseite gebildet werden kann. Zudem sollte betrachtet werden, wie detailliert die Klassen den Standard abbilden. Standards wie openTRANS® können umfangreicher sein als es für den konkreten Einsatz nötig wäre. Anfangs reicht es möglicherweise aus, den Anforderungen entsprechend nur die minimalst benötigten Eigenschaften des Standards zu implementieren. Sollte sich herausstellen, dass die Umsetzung doch umfangreicher werden soll, dann stellt eine flexible Softwarearchitektur sicher, dass sie ohne größeren Aufwand erweitert oder abgeändert werden kann. Ein Beispiel in dieser Arbeit ist das in Kapitel 4.1 beschriebene Fluent Interface.

5.2 Ausblick

Die ORDER und die ORDERRESPONSE sind nur zwei von zehn Geschäftsdokumenten, die durch den openTRANS®-Standard spezifiziert werden. In diesem Kontext wäre es denkbar und sinnvoll auch die Auftragsänderung (ORDERCHANGE) zu implementieren. Der Import einer Auftragsänderung würde es dem Kunden ermöglichen, zu einem gewissen Maß Änderungswünsche an einen vorher gesendeten Auftrag zu stellen.
Der Import von Aufträgen und der Export von Auftragsbestätigungen ist zweifelsfrei ein wichtiger Bestandteil, aber letztendlich ist es das Zusammenwirken von Auftrag, Auftragsänderung und Auftragsbestätigung, welches den Geschäftsprozess vollkommen macht. Eine Implementierung des Imports von Auftragsänderungen war für diese Arbeit nicht vorgesehen, sollte aber mit der Nutzung beziehungsweise Erweiterung der erstellten Softwarearchitektur mit nicht allzu hohem Aufwand möglich sein.

Quellenverzeichnis

Alpar, Paul; Alt, Rainer; Bensberg, Frank; Weimann, Peter (2019): Anwendungsorientierte Wirtschaftsinformatik. Strategische Planung, Entwicklung und Nutzung von Informationssystemen. 9., überarbeitete und aktualisierte Aufl. Wiesbaden: Springer Vieweg.

Balzert, Helmut (2009): Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering. 3. Aufl. Heidelberg: Spektrum Akademischer Verlag.

Broy, Manfred; Kuhrmann, Marco (2021): Einführung in die Softwaretechnik. Berlin: Springer Vieweg.

Engels, Barbara (2017): Bedeutung von Standards für die digitale Transformation. Befunde auf Basis des IW-Zukunftspanels. In: IW-Trends. 44. Jg. Nr. 2, Vorabversion. S. 21 – 40.

Fowler, Martin (2005): FluentInterface.
(https://martinfowler.com/bliki/FluentInterface.html, verfügbar am 24.10.2022)

Hoffmann, Dirk W. (2013): Software-Qualität. 2. Aufl. Berlin, Heidelberg: Springer Vieweg.

Ingenious Software GmbH (2022) (a): Das Unternehmen Ingenious Software GmbH
(https://ingenious.de/unternehmen/, verfügbar am 20.09.2022)

Ingenious Software GmbH (2022) (b): Übersicht der Funktionen der ingenious Branchensoftware
(https://ingenious.de/funktionsuebersicht/, verfügbar am 20.09.2022)

ISO/IEC (2011): ISO/IEC 25010:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models.

Jessen, Johann; Lenz, Barbara; Roos, Horst J.; Vogt, Walter (Hrsg.) (2003): B2C Elektronischer Handel - eine Inventur Unternehmensstrategien, logistische Konzepte und Wirkungen auf Stadt und Verkehr. Opladen: Leske + Budrich.

Koch, Frank (2003): Handbuch Software- und Datenbank-Recht. Berlin, Heidelberg: Springer.

Microsoft (2022) (a): Delegates (C# Programming Guide). (https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/, verfügbar am 24.10.2022)

Microsoft (2022) (b): Partial Classes and Methods (C# Programming Guide). (https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/partial-classes-and-methods, verfügbar am 19.11.2022)

Sandhaus, Gregor; Berg, Björn; Knott, Philip (2014): Hybride Softwareentwicklung. Das Beste aus klassischen und agilen Methoden in einem Modell vereint. Berlin, Heidelberg: Springer Vieweg.

Schmitz, Volker; Kelkar, Oliver; Otto, Boris; Weiner, Nico (2009) (a): openTRANS® Version 2.1. ORDER – Auftrag. (https://www.digital.iao.fraunhofer.de/de/publikationen/OpenTRANS21/Download-OpenTrans_V2_1.html, verfügbar am 19.11.2022).

Schmitz, Volker; Kelkar, Oliver; Otto, Boris; Weiner, Nico (2009) (b): openTRANS® Version 2.1. ORDERRESPONSE – Auftragsbestätigung. (https://www.digital.iao.fraunhofer.de/de/publikationen/OpenTRANS21/Download-OpenTrans_V2_1.html, verfügbar am 19.11.2022).

World Wide Web Consortium (W3C) (2008): Extensible Markup Language (XML) 1.0 (Fifth Edition). (https://www.w3.org/TR/REC-xml/, verfügbar am 14.09.2022).

World Wide Web Consortium (W3C) (2015): XML Essentials.

(https://www.w3.org/standards/xml/core, verfügbar am 14.09.2022)

Anlagen

Musterfirma
Hauptstraße 1
01234 Musterstadt

Kommission: **120087296**
Sachb.: Administrator
Kundennr.: 8041221108

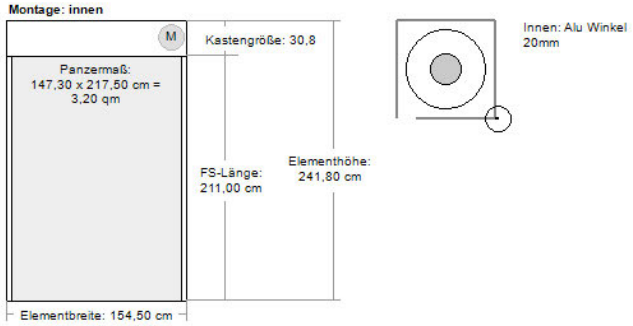
Auftragsbestätigung 202211/0121

Bestellung vom 18.11.2022

Ihr Auftrag wurde entsprechend der unten stehenden Auflistung bearbeitet. Bitte prüfen Sie alle Auftragsdaten wie Rollladentyp, Fertigmaße und Farben.
Eventuelle Änderungen sind innerhalb von 24 Stunden schriftlich zu melden!
Kurzfristige Aufträge (Vorlauf innerhalb von 4 Tagen vor dem Liefertermin) gehen direkt nach der Erfassung in den Produktionsablauf!

Pos.Nr.	Produkt/Abmessungen	Menge	EP €	GP €
001	**Revision von innen, für** **Klinkerbau mit verkürzter Außenschürze** Putzträger innen (Alu 20 mm) in weiß, Kunststoff-Panzer XL PVC, maxi Profil Z52, grau, 41 gelochte Stäbe, WEMX - Alu Winkel-Hohlschlussleiste für Neubauprofile mit Anschlag grau RAL7047 mit Endstabgleiter Endleiste ungebohrt inkl. Abrollprofil Veka 82mm in weiß Maxi-RF universal 59x47, weiß mit beidseitig grauer Bürste eingezogen und Abschlüsse in weiß 60-er Welle, Motor KAISER NIENHAUS Rohrmotor Primus Electronic Mercato Leistung: 15U/min., 113W / 10Nm, mit Auflaufschutz und aktivierbarer Hinderniserkennung, Parallelschaltung der Motoren möglich, elektronische Einstellung der Endlagen mit 3m-Anschlussleitung, inkl. Adapter/Mitnehmer für Welle SW60	1,00	311,00	311,00
	Materialteuerungszuschlag 12 %			*37,32*
	Mehrpreis für 3 Deckenbefestigungen	3,00	19,50	58,50
	Mehrpreis für Putzträgerprofil innen	1,00	13,29	13,29
	Mehrpreis für Führungsschienenabschlüsse	1,00	2,50	2,50
	Mehrpreis für Motor-Antrieb	1,00	153,00	153,00
	Mehrpreis für Endstab WEMX-Alu	1,00	12,05	12,05
	Mehrpreis für Klinkerbau mit verkürzter Außenschürze	1,00	11,00	11,00

Montage: innen



BLR-Maß 154,5 cm x 211,0 cm Antriebseite: rechts von innen
Windwiderstandsklasse WK3 wird mit Führungsschienentiefe 28 mm erreicht.

		Gesamt:	**598,66**
Zwischensumme			**598,66**
Teuerungszuschlag Artikelnummer: ***Menge: 1,00***	1,00	40,71	40,71
Nettosumme			**639,37**
zzgl. MwSt. 19 %			121,48
Gesamtbetrag			**760,85**

Zahlungsbedingung: Bankeinzug 10 Tage - 4% Skonto.

Bei einem Warenwert unter 100,00 € berechnen wir eine Frachtpauschale in Höhe von 20,00 €.
Bei Unstimmigkeiten wenden Sie sich bitte an den zuständigen Sachbearbeiter. Vielen Dank!

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Leipzig, den 26.11.2022

Edgar Schnurpel