Template

Christoph Hellmich

21. September 2018

Inhaltsverzeichnis

Über arc42

arc42, das Template zur Dokumentation von Software- und Systemarchitekturen.

Erstellt von Dr. Gernot Starke, Dr. Peter Hruschka und Mitwirkenden.

Template Revision: 7.0 DE (asciidoc-based), January 2017

Einführung und Ziele

Dieses Dokument beschreibt die Softwarearchitektur des AutoLiefer-Systems für das automatische Erfassen und Verarbeiten von Lieferscheinen für das Unternehmen CarLogistic AG.

Vision der FA. CarLogistic AG

Optimierung der Prozesse für die Buchhaltung, um langfristig die Qualität der Verwaltung zu garantieren (Agenda bis 2025).

Geschäftsziele

Z1 Steigerung der Effektivität bei der Überprüfung eingegangener Lieferscheine und Rechnungen durch die Senkung von manuellen Schritten sowie der Erhöhung der Teilautomatisierung. Erhoffte Vorteile:

- Senkung der Fehlerquote bei der Überprüfung von Lieferscheinen und Rechnungen
- Generelle Entlastung der zuständigen Mitarbeiter
- Skalierbarer Ansatz für die Zukunft

Z2 Integration von Techniken zur Auswertung und Bewertung von Vorgängen, die zu einer kontinuierlichen Optimierung des Prozesses führen.

Z3 Erhöhte Anpassbarkeit der Prozesse für zukünftige Anforderungen

Ausgangssituation

Abbildung 1 zeigt die Ist-Situation bei Projektbeginn als Paketdiagramm.

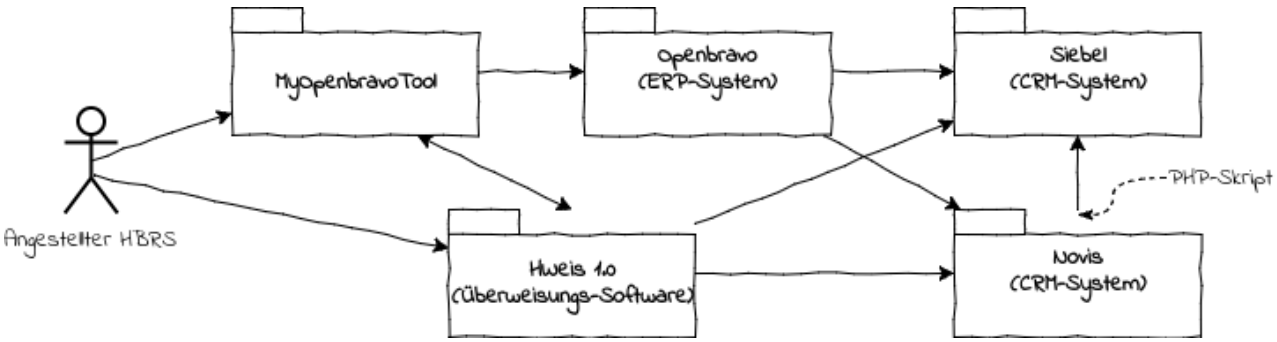

Abbildung 1: Ist-Situation

Die in Abbildung 1 dargestellten Systeme sind in Tabelle 1 beschrieben.

In Abbildung 2 ist ein möglicher Ablauf dargestellt, der bei einer Annahme einer Bestellung durchlaufen wurde.

Tabelle 1: Ist-Situation Systeme

System	Art	Beschreibung
HWeis 1.0	Überweisungssoftware	Rechnnungen überweisen
Openbravo	ERP-System	Posten und Betrag der Rechnung eintragen
MyOpenbravoTool	Desktop-Anwendung	Client-Tool für den Zugriff auf Openbravo
Siebel	CRM-System	Speicherung von Stammdaten und beweglichen Daten
Novis	CRM-System	Speicherung von Stammdaten und beweglichen Daten

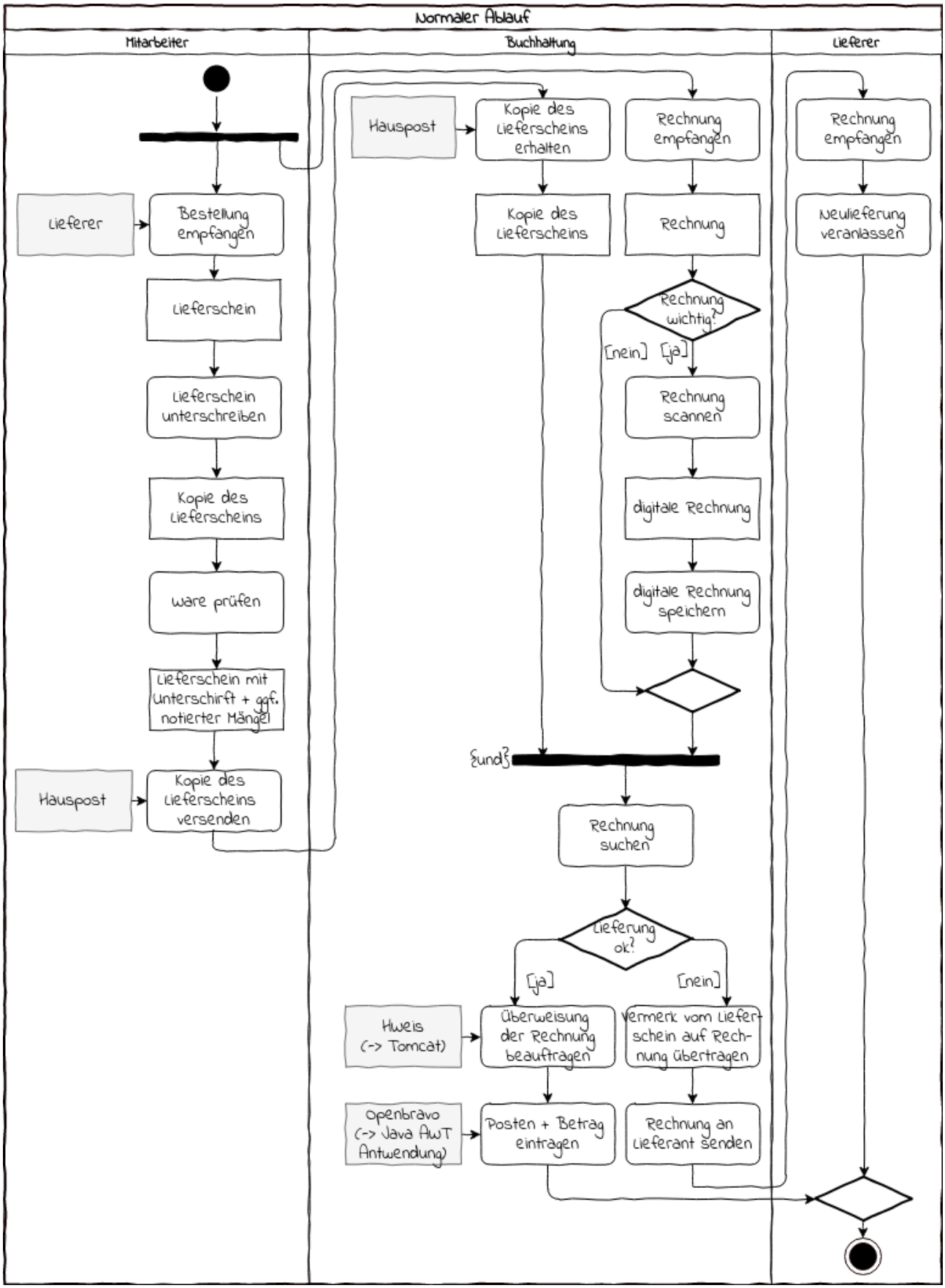

Abbildung 2: Aktuelles (As-Is) Szenario zur Orientierung (Standard Verlauf)

Aufgabenstellung

Anforderungen, die im Prototypen implementiert wurden, sind nachfolgend **fett** hervorgehoben. Auslassungen gegenüber den ursprünglichen Anforderungen sind ~~durchgestrichen~~ und Hinzufügungen stehen in [eckigen] Klammern. Anschließend sind in Abbildung 3 die Anforderungen in einem Use-Case-Diagramm zusammengefasst.

Funktionale Anforderungen (Sicht Auftraggeber = Mitarbeiter der CarLogistic)

FA1 Der Auftraggeber der Bestellung (Mitarbeiter der CarLogistic) soll in der Lage sein, Lieferscheine einscannen zu können.

FA2 Der Auftraggeber der Bestellung muss in der Lage sein, die wichtigsten Daten eines Lieferscheines manuell in einer Eingabemaske einzugeben, falls kein Scanner vorhanden ist.

FA3 Der Auftraggeber der Bestellung sollte in der Lage sein, einen eingescannten Lieferschein oder **einen manuell erfassten Lieferschein mit einer Anmerkung zu annotieren.**

FA4 Der Auftraggeber der Bestellung sollte in der Lage sein, einen eingescannten Lieferschein oder einen manuell erfassten Lieferschein mittels digitaler Signatur zu unterschreiben.

FA5 Die digital erfassten Lieferscheine müssen persistent in einem ECM gespeichert werden. Der importierte Lieferschein sollte dabei in einem gängigen Format abgespeichert werden. Das ECM befindet sich in der Abteilung Controlling.

FA6 ~~Die persistent abgespeicherten Lieferscheine müssen für spätere Bearbeitungen abrufbar sein.~~ [Redudant FA9]

FA7 Der Auftraggeber der Bestellung muss in der Lage sein, den eingescannten oder **manuell eingegebenen Lieferschein an die Buchhaltung der CarLogistic abzuschicken.**

FA8 Der Auftraggeber der Bestellung kann über den Status bei der weiteren Bearbeitung (z.B. Eskalation) durch das System stets unterrichtet werden. Dazu steht dem Auftraggeber eine entsprechende Web-Schnittstelle zur Verfügung.

FA9 Der Auftraggeber soll in der Lage sein, von Seiten der Buchhaltung der CarLogistic als falsch deklarierte Lieferscheine wieder zu empfangen und diese ggf. gemäß den Anmerkungen der Buchhaltung zu überarbeiten.

FA10 Der Auftraggeber kann sich Statistiken über seine abgeschlossenen Vorträge anzeigen lassen.

Funktionale Anforderungen (Sicht Mitarbeiter Buchhaltung der CarLogistic)

FA11 Der Rechnungsempfänger (ein ungeschulter Mitarbeiter Buchhaltung der CarLogistics, typischerweise eine studentische Hilfskraft) soll in der Lage sein, per Post eingegangene Rechnungen von Lieferanten einscannen zu können.

FA12 Die importierte Rechnung pro Lieferant sollte in einem gängigen Format in einem ECM archiviert werden.

FA13 Der Rechnungsempfänger muss in der Lage sein, die wichtigsten Daten einer Rechnung manuell einzugeben, falls kein Scanner vorhanden ist.

FA14 Der Rechnungsempfänger muss in der Lage sein, elektronisch eingegangene Rechnungen vom Lieferanten zu empfangen und über dieses Ereignis entsprechend unterrichtet zu werden.

FA15 Der Rechnungsempfänger sollte in der Lage sein, eine eingescannte, **manuell erfasste** oder elektronisch empfangene **Rechnung mit einer Anmerkung zu annotieren.**

FA16 Der Rechnungsempfänger muss in der Lage sein, eine eingescannte, **manuell erfasste** oder elektronisch empfangene **Rechnung zur weiteren Bearbeitung an den Rechnungsprüfer weiterzuschicken.**

FA17 ~~Der Rechnungsempfänger möchte die Eskalation von Rechnungen und Lieferscheinen vornehmen (inkl. Kommentar). Der Einkäufer wird dabei in bestimmten Fällen unmittelbar unterrichtet, um die Reaktionszeit auf stark eskalierte Vorgänge zu reduzieren. Eine Eskalation erfolgt bei einfachen bis zu gravierenden Abweichungen in der Lieferung und der Rechnungsstellung. Eine Eskalation hängt auch von dem betreffenden Unternehmen ab sowie von der Anzahl vergangener Eskalationen.~~ [Widerspruch FA27]

FA18 ~~Für Statistikzwecke müssen die eingegangen Rechnungen pro Lieferant persistent gespeichert werden.~~ [Redudant FA12] ~~Zudem müssen die importierten Lieferscheine der Auftraggeber persistent gespeichert werden. Der importierte Lieferschein sollte dabei in einem gängigen Format abgespeichert werden.~~[Redudant FA5] ~~Hierzu sollte das ECM der Abteilung Marketing verwendet werden.~~ [Widerspruch FA5]

FA19 Der Rechnungsprüfer (Mitarbeiter in der Buchhaltung der CarLogistic, typischerweise ein erfahrener Buchalter mit mehrjähriger Erfahrung) wird vom System regelmäßig über eingehende Lieferscheine (von Seiten des Auftraggebers) ~~unterrichtet (Email-Notifikation oder Signal in der Client-Applikation)~~**, [mittels eines neuen Eintrags in seiner personalisierten Task-Liste innerhalb der Client-Anwendung unterrichtet, vgl. FA20 und FA21.]**

FA20 Der Rechnungsprüfer wird vom System regelmäßig über eingehende Rechnungen unterrichtet. Dazu erhält der Rechnungsprüfer entsprechend einen neuen Eintrag in seiner personalisierten Task-Liste innerhalb der Client-Anwendung.

FA21 Das System soll ~~**identische**~~ **[zusammengehörige] Rechnungen und Lieferscheine identifizieren und ein Zusammenhang dem Rechnungsprüfer vorschlagen und in seiner personalisierten Task-Liste ablegen.** ~~Eine Bestätigung des Zusammenhangs~~ **[Die Zuordnung des Systems] muss der Rechnungsprüfer bestätigen.**

FA22 Der Rechnungsprüfer soll in der Lage sein, Bezüge zwischen Rechnungen und Lieferscheinen manuell herzustellen, falls das System keine Bezüge hat herstellen können.

FA23 Falls der Lieferschein[, dem eine Rechnung zugewiesen wurde,] eine Lieferung bestätigt, so kann die Bearbeitung des Rechnungsbetrags durchgeführt werden. Diese Bestätigung wird in dem Vorgang persistent gespeichert (Datum, Kommentar usw.)

FA24 Die Banküberweisung des Rechnungsbetrags an die Bank wird automatisiert durch das System ausgelöst.

FA25 Die Buchung des Rechnungsbetrags in die Finanzbuchhaltung des ERP-Systems soll automatisiert durch das System erfolgen.

FA26 Kann die Überweisung wegen einer fehlerhaften Lieferung nicht ausgeführt werden, so ist die Rechnung dem Lieferanten digital oder per Post zurückzuschicken. Der Vermerk des Mitarbeiters aus dem Lieferschein soll dabei verlustfrei übernommen werden. Diese Ablehnung wird in dem Vorgang persistent gespeichert (Datum, Kommentar).

FA27 Der Rechnungsprüfer muss die Eskalation von Rechnungen und Lieferscheinen initiieren können. Bei einer Eskalation soll er zunächst einen erklärenden Kommentar eingeben. Der Einkäufer, in kritische Fällen aber auch andere Stakeholder (z.B. Einkäufer, Vorstand, Präsident) werden dabei in bestimmten Fällen unmittelbar anhand verschiedener Eskalationsstufen unterrichtet, um die Reaktionszeit auf stark eskalierte Vorgänge zu reduzieren. Nur der Rechnungsprüfer darf Eskalationen vornehmen.

Stufe 1 Differenz Rechnung und Lieferschein < 100: Einkäufer informieren

Stufe 2 Differenz Rechnung und Lieferschein zwischen 100 und 1000: Vorstand informieren

Stufe 3 Differenz Rechnung und Lieferschein > 1000: Präsident informieren

FA28 Der Rechnungsprüfer kann nicht korrekt ausgefüllte Lieferscheine an den Auftraggeber zurücksenden, mit der Bitte um Überarbeitung (inkl. Kommentar).

FA29 Bei der Kontrolle von stark abweichenden Rechnungen und Lieferscheinen (gemäß Eskalationsstufen) gilt das Vier-Augen-Prinzip.

FA30 Bei vertrauenswürdigen Lieferanten kann der Prozess (Abgleich elektronisch eingegangene Rechnung mit Lieferschein, Verbuchung, Überweisung, Archivierung) automatisiert ablaufen.

Funktionale Anforderungen (Einkäufer der CarLogistic)

Anmerkung: der Einkäufer ist eine Rolle in der Verwaltung, der für den Einkauf sämtlicher Güter und Dienstleistungen zuständig ist.

FA31 Der Einkäufer des Unternehmens soll über ein Cockpit die fachliche Leistungsfähigkeit des neuen Prozesses mit Hilfe von Key Performance Indicators (KPI) regelmäßig unterrichtet werden.

- **KPI1** Wie oft werden Rechnungen an einen Lieferer zurückgeschickt?
- **KPI2** Wie oft wird von welchem Lieferer bestellt?
- **KPI3** Wie oft hat das System automatisch und korrekt einen Bezug zwischen Lieferschein und Rechnung hergestellt?
- **KPI4** Wie oft hat das System einen falschen Bezug zwischen Lieferschein und Rechnung hergestellt?
- **KPI5** Wie oft wurde welche Eskalationsstufe erreicht?

FA32 Der Einkäufer soll über alle abgeschlossenen Vorgänge (definiert von der eigentlichen Bestellung bis zur Rechnungsüberweisung) wöchentlich vom System in Form eines Reports unterrichtet werden.

FA33 Der Einkäufer nimmt Eskalationen von Seiten der Rechnungsprüfung täglich in Form eines Reports entgegen. Der Einkäufer kommentiert diese (Kommentar wird zum Vorgang gespeichert) und entscheidet ad hoc über den weiteren Vorgang (→ separater Fall mit verschiedenen Handlungsoptionen, die er situativ auswählen kann). Eine Eskalation wird nur intern verarbeitet, eine Weiterleitung an den Lieferanten ist nicht vorgesehen.

Funktionale Anforderungen (Lieferant)

FA34 Der Lieferant soll über eine Schnittstelle, elektronische Rechnungen an die CarLogistics versenden können. Die Erstellung der Rechnung soll nicht Bestandteil des Systems sein.

FA35 ~~Zur Förderung der besseren Zusammenarbeit und Zuverlässigkeit möchte der Lieferant bei hoch eingestuften Eskalationen unmittelbar vom System unterrichtet werden.~~ [Widerspruch FA33]

Technische Anforderungen (gestellt von der IT-Abteilung)

TA1 Die technische Abteilung wünscht sich eine zentrale Kommunikation über eine Komponente, um die Kommunikation in der Software-Architektur besser überwachen und loggen zu können. Der technischen Abteilung ist die Komplexität einer solchen Lösung durchaus bewusst und möchte eine Einschätzung hinsichtlich der Machbarkeit / Rentabilität haben. [ESB]

TA2 Die technische Abteilung möchte eine zentrale Registry zur Verwaltung der implementierten Services haben. [Service Registry]

TA3 Die technische Abteilung wünscht sich, auch in fachlicher Abstimmung mit dem Einkäufer, eine flexible Art und Weise, um Eskalationsregeln ändern zu können. Auch über die Darstellung von Eskalationsregeln möchte die technische Abteilung eine Information haben. [BRE]

TA4 Die technische Abteilung soll über ein Cockpit die technische Leistungsfähigkeit des neuen Prozesses mit Hilfe von Key Performance Indicators (KPI) regelmäßig unterrichtet werden. [BAM]

TA5 Die technische Abteilung soll in der Lage sein, wichtige Abläufe in der Architektur als Workflow zu ändern und in der Umgebung einzusetzen. Damit sollen zudem auch zukünftig neue Geschäftsprozesse flexibel und agil unterstützt werden. [BPM/BPMN]

TA6 Die technische Abteilung wünscht sich nachvollziehbare Abkommen zwischen den Lieferanten, der CarLogistics, um etwaige Datenvolumen, Zugriffe auf Systeme zu regulieren. Diese Abkommen sollen auf belastbaren Annahmen basieren. [SLA]

TA7 Die technische Abteilung wünscht sich einen State-Of-The-Art Ansatz zur Entwicklung der notwendigen Client-Anwendung inklusive der Umsetzung von personalisierten Task-Listen.

Die Anwendung sollte möglichst einheitlich sein, um ein Corporate Design zu realisieren. [Web-User-Interface]

TA8 Die technische Abteilung möchte eine technische Unterstützung haben zur Umsetzung der Eskalation seitens des Einkäufers (FA33). Er möchte einen State of the Art Ansatz haben, wie man ad hoc ausführbare Handlungsoptionen modellieren und implementieren kann. [BPM/CMMN]

TA9 Die technische Abteilung möchte zur Kostenreduktion wichtige Bestandteile der Architektur in eine Cloud auslagern und ist hierzu auch gerne an Lösungsstrategien interessiert. [**On-Premise** vs. Cloud]

TA10 Die technische Abteilung möchte generell die Skalierbarkeit sowie die Ausfallsicherheit der Gesamt-Architektur optimieren, damit auch in Zukunft mit einer steigenden Anzahl von Benutzern und Daten umgegangen werden kann. [Skalierbarkeit und Ausfallsicherheit, nur modellieren]

Forschungsperspektiven (Prof. Mayer, Leiter Innovationsabteilung)

FF1 Ein Professor wünscht sich ein Modell zur Empfehlung einer Handlungsoption, die dem Einkäufer die bestmögliche Variante präsentiert, die er dann auswählen kann (betr. FA33). Eine prototypische Implementierung wäre auch wünschenswert.

Qualitätsziele

Das primäre Qualitätsziel für das vorliegende Projekt ergibt sich aus der technischen Anforderung 3 (TA3). Die Eskalationsregeln sollen komfortabel bearbeitet werden können. Für die Verwaltung der Eskalationsregeln wird eine Business Rules Engine (BRE) verwendet.

Des Weiteren soll das System nach der technischen Anforderung 10 (TA10) skalierbar sein.

Weitere Qualitätsziele für die Entwicklung des Prototypen sind:

Verständlichkeit Der Code soll einfach zu verstehen und der Lösungsweg leicht zu adaptieren sein.

Funktionalität Der Prototyp soll eine möglichst hohe Zahl der Anforderungen erfüllen.

Änderbarkeit Es sollten nur die nötigsten Entwurfsentscheidungen getroffen werden. Zwischen einzelnen Entscheidungen sollte möglichst keine Abhängigkeit bestehen.

Anpassbarkeit Der Prototyp sollte offen für Ergänzungen und Erweiterungen sein. Eine hohe Testabdeckung könnte dies beispielsweise erschweren.

Stakeholder

Die Tabelle 2 zeigt die konkreten Stakeholder für das System sowie deren Erwartungshaltung.

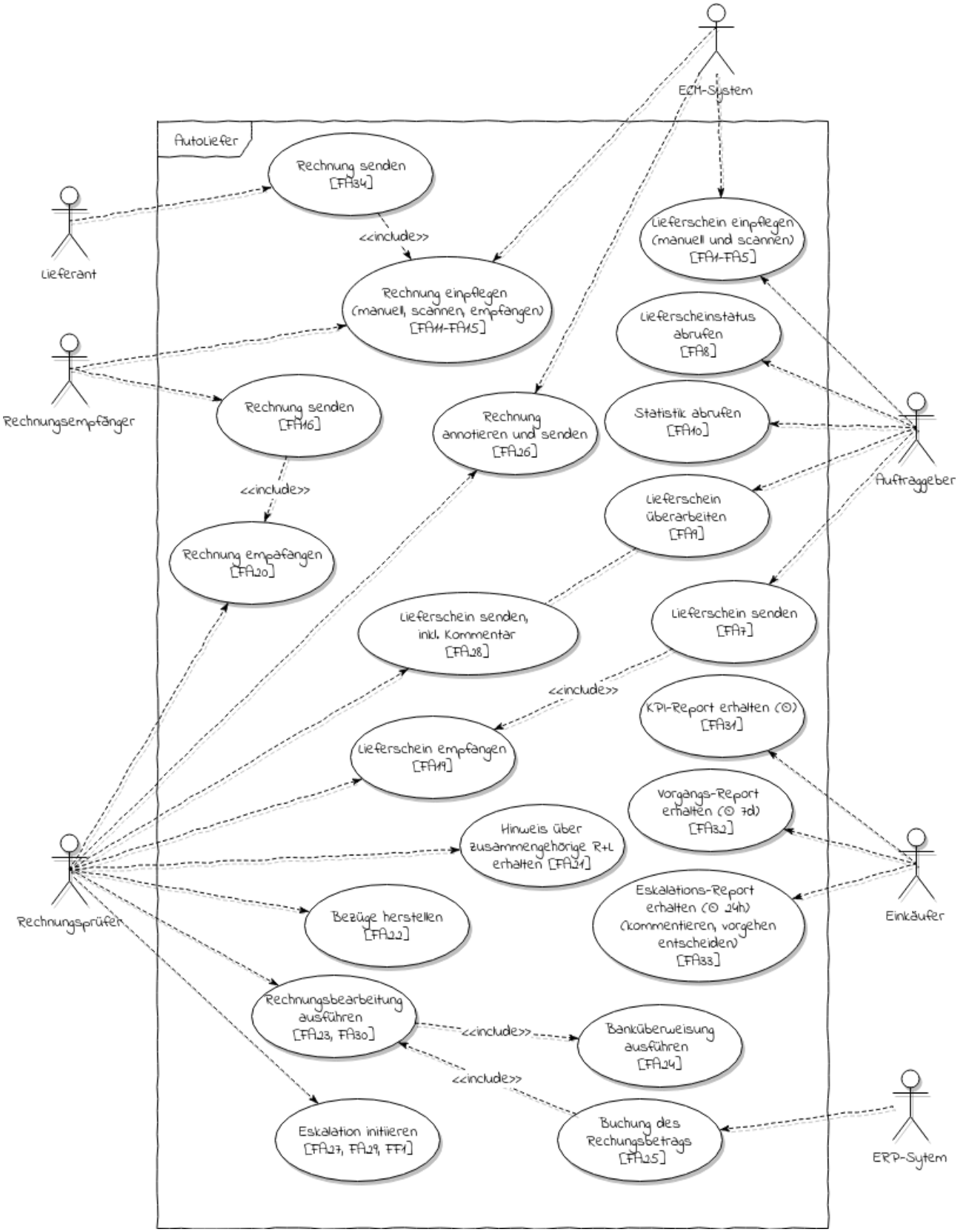

Abbildung 3: Use-Case-Diagramm

Tabelle 2: Stakeholder

Rolle	Beschreibung	Erwartungshaltung
Auftraggeber	Mitarbeiter der CarLogistic; erhält Lieferschein	Entlastung
Rechnungsempfänger	Mitarbeiter aus der Buchhaltung; erhält Rechung	Unterstützung bei der Einpflege von Rechnungen in das System
Rechnungsprüfer	Mitarbeiter aus der Buchhaltung; verarbeitet Rechung und Lieferschein	Ünterstützung bei der Zuordnung von Lieferscheinen und Bestellungen
Einkäufer	Mitarbeiter aus der Verwaltung; überwacht Bestellung	Benachrichtigung, sobald Probleme mit einer Bestellung auftreten; Überblick über getätigte Bestellungen.
IT-Abteilung	Wartet das zu entwickelnde System	Leichte Anpassbarkeit und Überwachung des Geschäftsprozesses
Lieferant	Bringt die Bestellung zum Auftraggeber in die CarLogistic	Schnelle Bezahlung des Rechungsbetrags
Prof. Mayer	Leiter der Innovationsabteilung	Erkenntnisgewinn

Randbedingungen

In diesem Abschnitt sind die Randbedingungen für das AutoLiefer-System beschrieben.

Technische Randbedingungen

Betrieb der Lösung auf einem marktüblichen Standard-Notebook, um sie im Rahmen einer Präsentation oder evtl. als Referenz (Bewerbungen etc.) auf einem solchen vorstellen bzw. besprechen zu können.

Für die Ausführung der Anwendung wird Java ab Version 8 und ein Web-Browser benötigt. Die Anwendung kann von der Kommandozeile aus gestartet werden.

Für das Projekt steht kein Budget zur Verfügung. Es können nur kostenlose Bibliotheken und Services verwendet werden.

Organisatorische Randbedingungen

Folgende Service Level Agreements sind einzuhalten:

SLA1 Rechnungen dürfen eine Größe von 12 MB nicht überschreiten.

SLA2 Alle eingehenden Rechnungen werden innerhalb von 7 Tagen verarbeitet.

SLA3 Der Service, um Rechnungen zu übertragen, steht 24 Stunden am Tag, 7 Tage die Woche mit einer Ausfallrate von maximal 0,1% im Jahr zur Verfügung.

Des Weiteren soll der Prototyp bis zum 12. September fertiggestellt werden. Die Dokumentation soll bis zum 16. September den finalen Zustand erreichen.

Kontext

Fachlicher Kontext

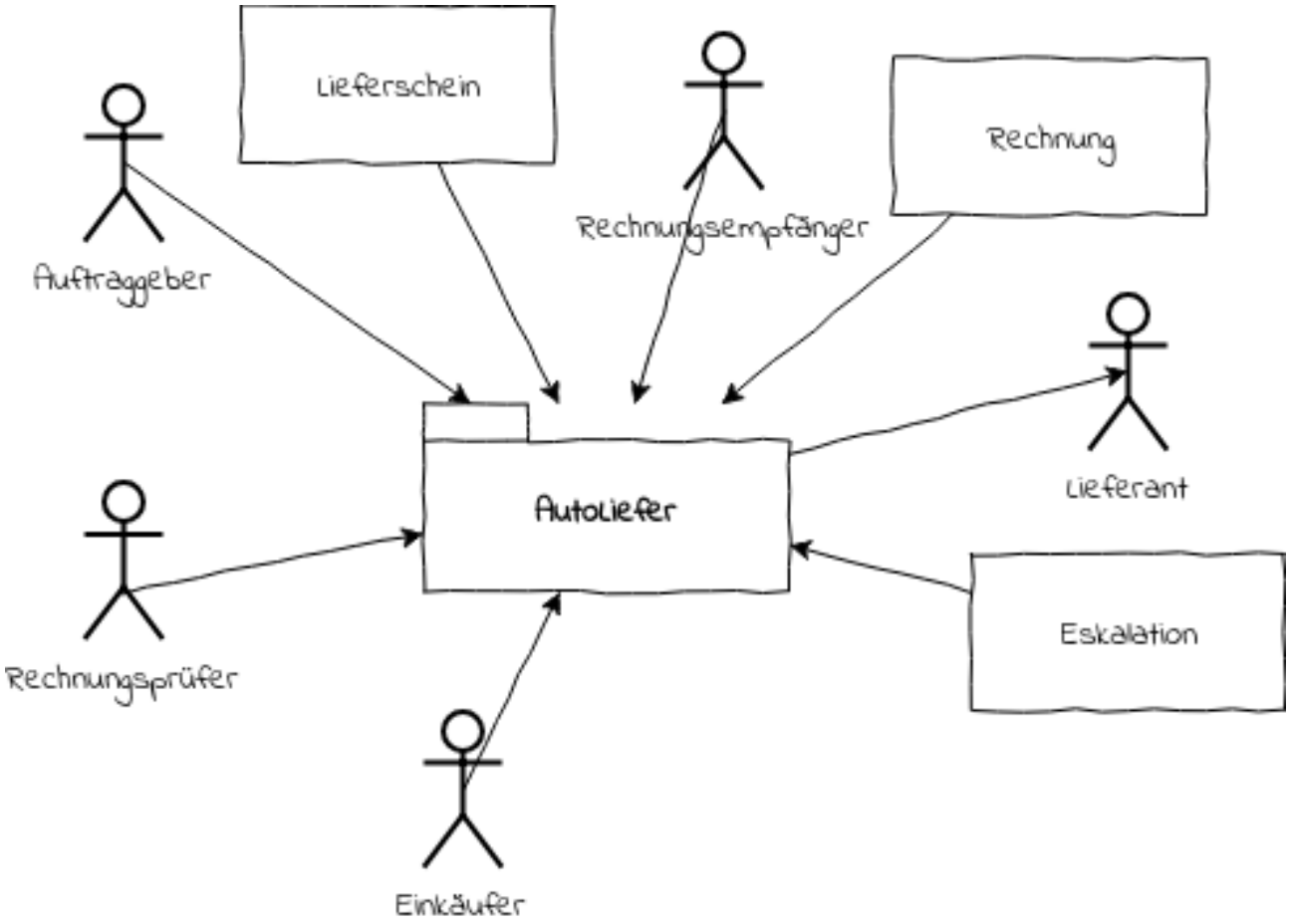

Abbildung 4: Fachlicher Kontext

Anwender

Anwender des Systems AutoLiefer sind Auftraggeber, Rechnungsprüfer, Rechnungsempfänger, Einkäufer und Lieferant, vgl. Stakeholder.

Lieferschein

Der Lieferschein wird vom Auftraggeber entgegengenommen, unterschrieben und in das System eingepflegt. Der Rechnungsprüfer bzw. das System versucht, zu jedem Lieferschein die zugehörige Rechnung zu finden. Stellt der Rechnungsprüfer Fehler hinsichtlich des Lieferscheins fest, wird dieser inklusive Kommentar, seitens des Rechnungsprüfers, an den Auftraggeber zur Korrektur zurückgesendet.

Rechnung

Eine Rechnung wird vom Lieferanten ausgestellt. Der Rechnungsprüfer erhält die Rechnung und pflegt diese, ggf. mit einem Kommentar, in das System ein. Der Rechnungsprüfer kontrolliert die Rechnung. Falls bei der Prüfung Mängel festgestellt wurden, wird die Rechnung inklusive eines Kommentars an den Lieferanten gesendet. Andernfalls, bei einer erfolgreichen Prüfung, wird eine Banküberweisung ausgeführt.

Eskalationen

Der Rechnungsprüfer initiiert Eskalationen von Rechnungen und Lieferscheinen. Eine Eskalation erfolgt bei einfachen bis zu gravierenden Abweichungen in der Lieferung und der Rechnungsstellung.

Technische Kontext

In Abbildung 5 ist der technische Kontext des AutoLiefer-Systems beschrieben.

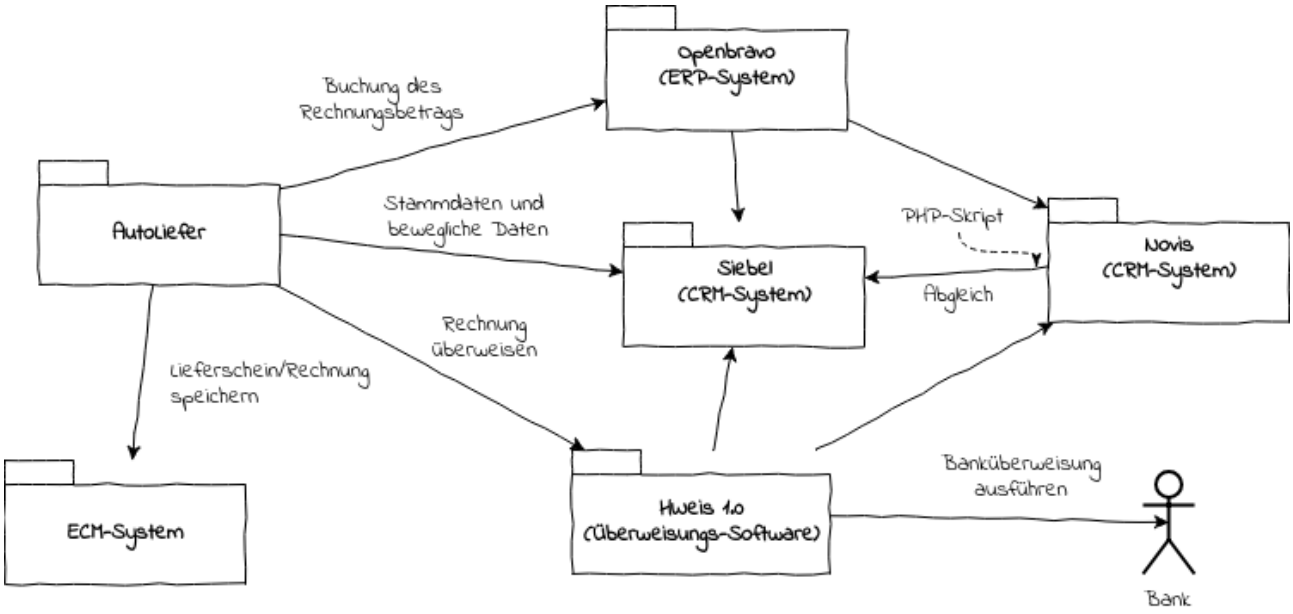

Abbildung 5: Technisher Kontext

ERP-System: OpenBravo

Der eingegangene Posten und der Betrag der überwiesenen Rechnung wird in der Finanzbuchhaltung eingetragen. Dies erfolgt manuell in dem ERP-System Openbravo.

Überweisungs-System: HWeis 1.0

Falls der Lieferschein den ordnungsgemäßen Empfang der Ware bestätigt, wird die Überweisung der Rechnung beantragt. Diese Überweisung erfolgt manuell über eine eigen entwickelte Überweisungssoftware (HWeis 1.0), einer internen Web-Anwendung, welche als Webserver (Tomcat 7) bereitgestellt wird.

Der Aufruf von HWeis erfolgt über Java Servlet-Schnittstelle. Es werden sowohl Anfragen mittels GET als auch POST unterstützt, um Rechnungen zu überweisen, zu speichern und zu löschen. Außerdem können Reports für die Quartalsabrechnung erzeugt, Stammdaten der Mitarbeiter verifiziert und die Raumplanung der Mitarbeiter vorgenommen werden.

Falls eine Rechnung überwiesen wurde, dann erfolgt ein Signal via RMI (an die Client-Anwendung MyOpenbravo-Tools), dass die Rechnung verbucht werden konnte. Im Fall einer Verbuchung, erfolgt ein Signal von OpenBravo zurück zu HWeis 1.0 zur „Fertigstellung des Vorgangs“.

CRM-Systeme: Novis und Siebel

Stammdaten und bewegliche Daten werden in zwei verschiedenen CRM-Systemen (Siebel und die Eigenentwicklung „Novis“) festgehalten, die vom Bereich CRM betrieben werden. Beide CRM-Systeme werden von OpenBravo und HWeis verwendet. Beide CRM-Systeme sind stets vom Datenbestand her konsistent abgeglichen.

Dies wird durch eine PHP-Skript (abgeschlossene Bachelor Thesis im FB Informatik) innerhalb der Software Novis realisiert, die wöchentlich die Daten zwischen den CRMs abgleicht.

ECM-System

In dem ECM-System werden Lieferscheine und Rechnungen gespeichert.

Lösungsstrategie

Für die Entwicklung des Prototypen liegt der Fokus auf der Verwendung von Funktionalität bestehender Lösungen, um von Best Practice erfahrener Entwickler profitieren zu können. Anschließend wird situativ entschieden, wo die Entwicklung eigener Funktionalität nötig ist.

Die Camunda Plattform enthält viele Funktionen, die für die Umsetzung der Anforderungen des AutoLiefer-Sytems aufgegriffen werden können. Die Modellierung der Workflows und der Entscheidungstabellen erfolgt mit dem Camunda Modeler, so dass die Bearbeitung von Abläufen bzw. der Regeln auch ohne Programmierkenntnisse möglich sind. Die Steuerung der Workflows und die Auswertung der Entscheidungstabellen übernimmt die Camunda Workflow Engine bzw. Camunda Decision Engine. Die Anzeige und Bearbeitung von Tasks wird mit der Camunda Taskslist realisiert, um zu vermeiden, eigenständig eine Benutzerverwaltung für den Prototypen implementiert zu müssen. Der Status der Workflows kann mit dem Camunda Cockpit abgerufen werden. Reports müssen selber erstellt werden, da die Funktionen nur in der kostenpflichtigen Enterprise Version von Camunda angeboten wird.

Die Benutzerfreundlichkeit und Fehlerbehandlung hat für die Umsetzung des Prototypen niedrige Priorität.

Bausteinsicht

Whitebox selektierten Ausschnitt des Systems

Die Bausteinsicht ist in Abbildung 6 abgebildet. Die Darstellung der Komponenten und Services sollte gemäß der Referenzarchitektur einer SOA nach Humm und Hess erfolgen. In den Tabellen 4, 5, 6 und 7 sind die Methoden der Services beschrieben.

Rechnungsverwaltung

Tabelle 4: Methoden Rechnungsverwaltung

Methode	Beschreibung
+ RechnungHinzufügen	Eine Rechnung erstellen
+ RechnungSenden	Eine Rechnung als Message versenden

Lieferscheinverwaltung

Tabelle 5: Methoden Lieferscheinssverwaltung

Methode	Beschreibung
+ LieferscheinHinzufügen	Ein Lieferschein erstellen
+ LieferscheinSenden	Ein Lieferschein als Message versenden

Zuordnung

Tabelle 6: Methoden Zuordnung

Methode	Beschreibung
+ ÜbereinstimmungenFinden	Übereinstimmende Rechnung oder Lieferschein suchen
+ ZuordnungSpeichern	Eine Zuordnung speichern
+ BenachrichtigungEskalation	Stakeholder über Eskalation benachrichtigen

Finanzverwaltung

Tabelle 7: Methoden Finanzverwaltung

Methode	Beschreibung
+ Überweisung ausführen	Buchung des Rechnungsbetrags mit OpenBravo
+ Rechnung buchen	Rechnung mit HWeis überweisen

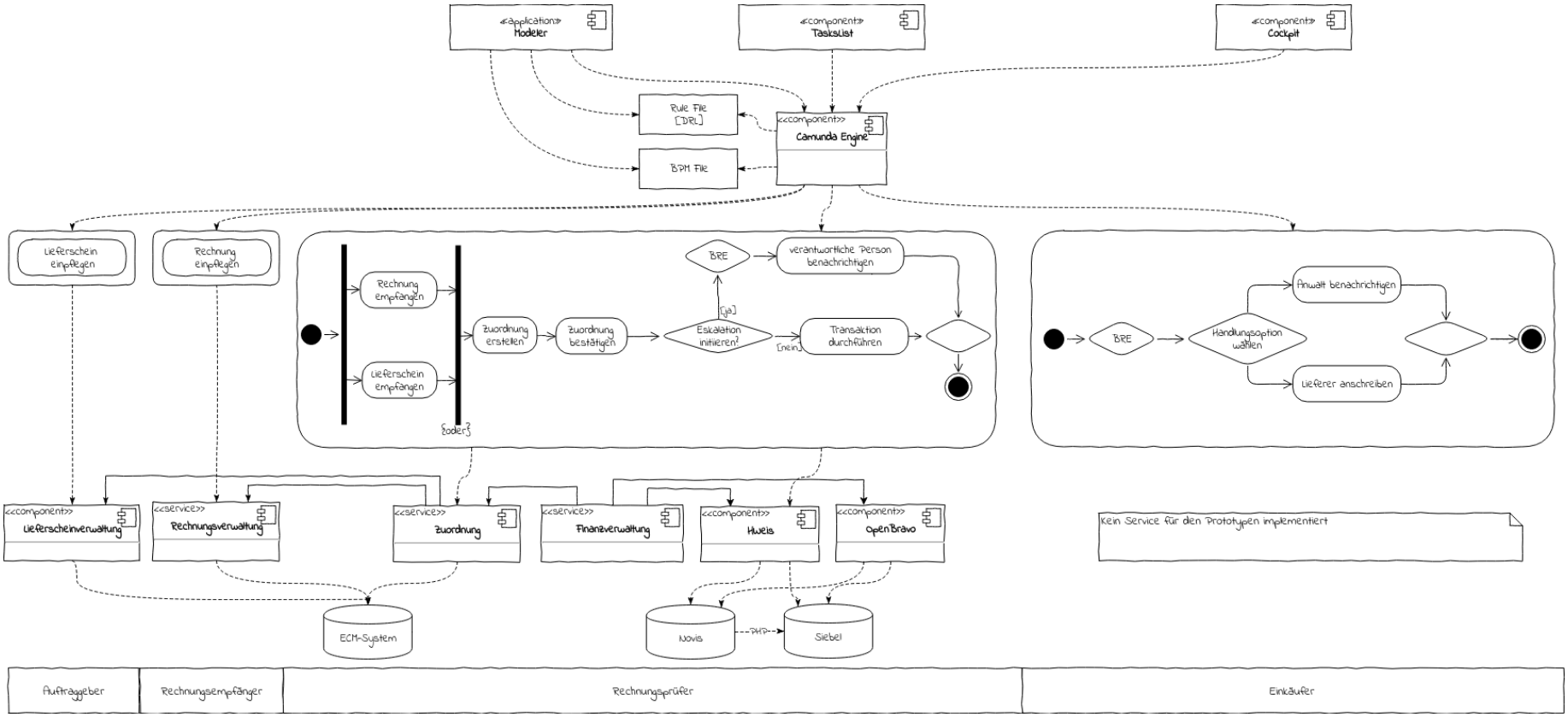

Abbildung 6: Bausteinsicht

Kommunikation der Services mit der Camunda Engine

Die Services kommunizieren mit der Camunda Engine mittels REST-Abfragen. Für das Senden von REST-Abfragen wurde die Bibliothek *Retrofit* verwendet. Retrofit nutzt das Interface *CamundaServiceApi*, siehe Listing 1, um die dort spezifizierten REST-Abfragen zu generieren. Die Services können über das Singleton *CamundaService*, siehe Listing 2, auf die Methoden zugreifen. Die CamundaService Instanz delegiert die Methodenaufrufe an das von Retrofit erzeugte Objekt.

Listing 1: CamundaServiceApi

```
internal interface CamundaServiceApi {
    @GET("process-definition")
    val allProcessDefinitions: Call<List<ProcessDefinition>>

    @POST("message")
    @Headers("Content-Type: application/json")
    fun sendMessage(@Body body: MessageBody): Call<ResponseBody>
}
```

In Abbildung 7 ist das Klassendiagramm zu den Listenings 1 und 2 dargestellt.

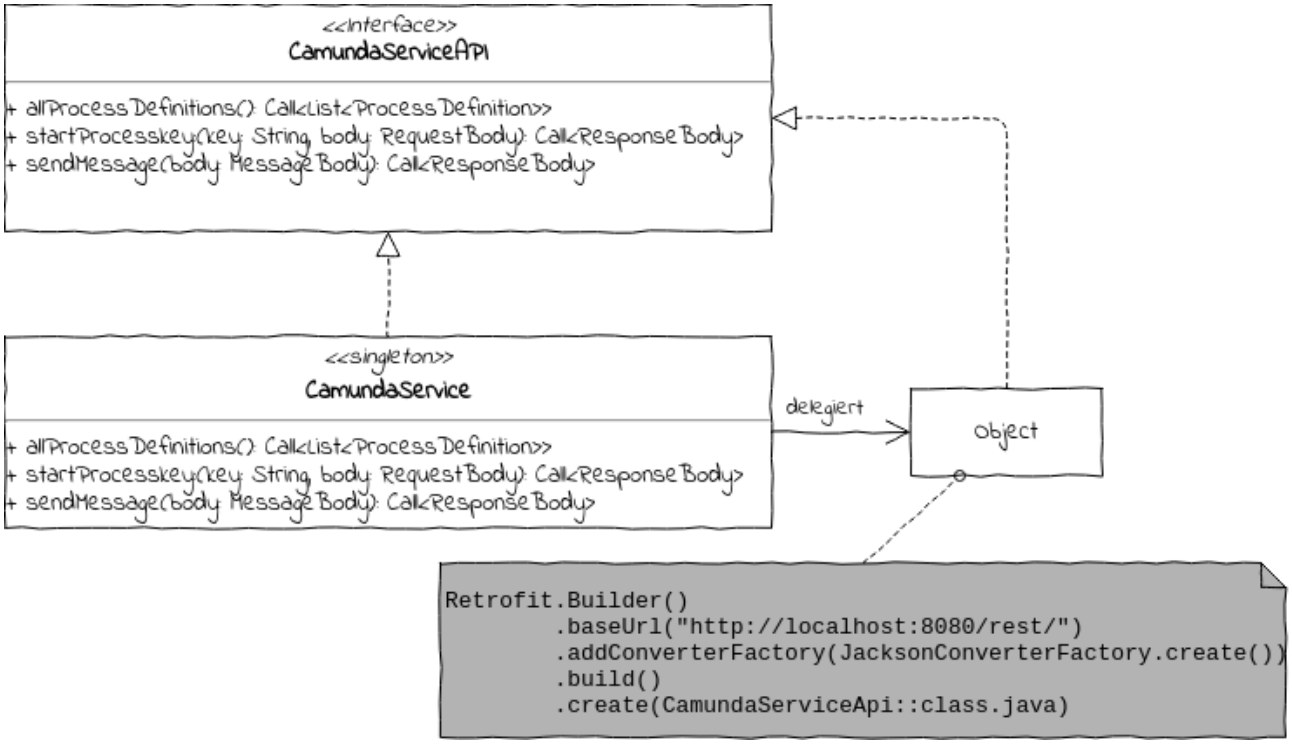

Abbildung 7: Kommunikation mit Camunda über Retrofit Bibliothek

Listing 2: CamundaService

```
object CamundaService : CamundaServiceApi by Retrofit
        .Builder()
        .baseUrl("http://localhost:8080/rest/")
        .addConverterFactory(JacksonConverterFactory.create())
        .build()
        .create(CamundaServiceApi::class.java)
```

In Listing 3 ist beispielhaft ein externer Tasks implementiert, der ein Lieferschein an eine bestimmte Adresse schickt.

Listing 3: Beispielimplementierung eines externen Tasks

```
// Erstelle ein von Camunda bereitgestelltes ExternalTaskClient-Objekt
val client = ExternalTaskClient.create()
        .baseUrl("http://localhost:8080/rest")
        .build()

// Warte auf Arbeit fuer das Topic "send-delivery-note"
//  (Das Topic ist im BPM-Modell definiert)
client.subscribe("send-delivery-note")
        .lockDuration(1_000)
        .handler { externalTask, externalTaskService ->
            // Extrahieren den Name der Nachricht aus dem Task
            val messageName = externalTask.getVariable<String>("messageName")

            // Erstelle deliveryNote aus den anderen Variablen des Tasks
            val deliveryNote = DeliveryNote(externalTask)

            // Gebe die Variablen der deliveryNote als eine Map aus
            val processVariables = deliveryNote.asProcessVariables()

            // Erstelle den Body der REST-Abfrage
            val body = MessageBody(messageName, processVariables)

            // Sende REST-Request
            //  POST http://localhost:8080/rest/message
            //  mit dem Content-Type: application/json
            //  und den zuvor erstellten body.
            CamundaService.sendMessage(body).execute() // s. Listing 1 und 2

            // Beende den Task
            externalTaskService.complete(externalTask)
        }.open()
```

Laufzeitsicht

In Abbildung 8 ist der Bestellprozess als BPM-Diagramm abgebildet. Das Digitalisieren des Lieferscheins und das Schreiben eines Kommentars erfolgen über ein Portal. Das Senden des Lieferscheins an den Rechnungsprüfer erfolgt vollständig automatisiert. Lehnt der Rechnungsprüfer einen Lieferschein ab, kann der Auftraggeber den abgelehnten Lieferschein einschließlich der Anmerkungen des Rechnungsprüfers anzeigen lassen und überarbeiten. Anschließend wird der überarbeitete Lieferschein vollständig automatisiert an den Rechnungsprüfer gesendet.

Wie bei dem Workflow für das Einpflegen des Lieferscheins, erfolgt auch das Digitalisieren der Rechnung über das Portal. Das Senden der Rechnung an den Rechnungsprüfer erfolgt ebenfalls vollständig automatisiert.

Werden bei dem Lieferschein Mängel festgestellt, kann der Rechnungsprüfer über das Portal die Mängel in einem Kommentar beschreiben und anschließend vollständig automatisiert den Lieferschein einschließlich des Kommentars an den Auftraggeber zurücksenden. Anschließend werden die Bezüge zwischen den eingegangenen Rechnungen und den erfolgreich geprüften Lieferscheinen vollständig automatisiert hergestellt. Kann das System keine Zusammengehörigkeiten zwischen Rechnung in Lieferschein identifizieren, muss der Rechnungsprüfer die Zuordnung über ein Portal manuell durchführen (im Prototypen nicht implementiert). Andernfalls überprüft der Rechnungsprüfer die vom System durchgeführte Zuordnung. Falls der Rechnungsprüfer die Zuordnung bestätigt, kann er ggf. eine Eskalation initiieren. Initiiert der Rechnungsprüfer eine Eskalation, ermittelt die BRE den zu benachrichtigen Stakeholder. Liegen keine Fehler vor, kann die Transaktion der Rechnung automatisiert erfolgen.

Der Einkäufer wird über Eskalationen in seiner Taskliste informiert. Die BRE schlägt dem Einkäufer eine mögliche Handlungsoption vor. Anschließend kann er über das Portal eine Handlungsoption auswählen, zu der Eskalation einen Kommentar verfassen und die Eskalation ausführen.

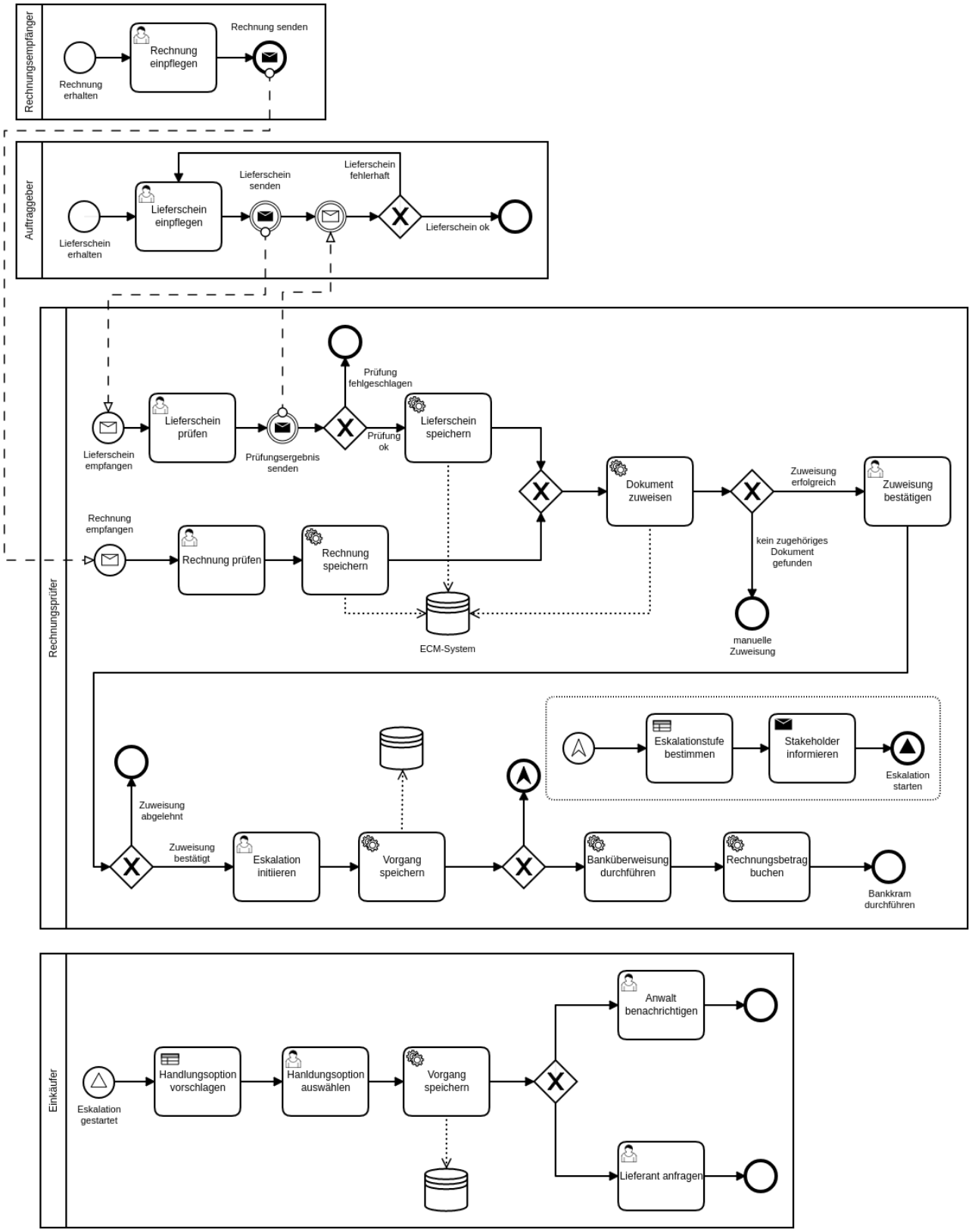

Abbildung 8: Bestellprozess

Verteilungssicht

Abbildung 9 zeigt die Verteilungssicht für den selektierten Ausschnitt des Systems (ausgegraute Elemente sind nicht Teil des Prototypen). Die Rechnungsverwaltung, die Lieferscheinverwaltung, der Zuordnungsservice und die Finanzverwaltung sind eigenständige Java-Applikationen, die über HTTP/REST mit dem Server kommunizieren. Bei dem Server handelt es sich um eine embedded Tomcat Instanz, die von SpringBoot gestartet wird und die Camunda-Engine ausführt. Die Entscheidungstabellen und die BPM-Workflows können mit der Desktop-Anwendung Camunda Modeler entwickelt und über HTTP auf den Server deployed werden.

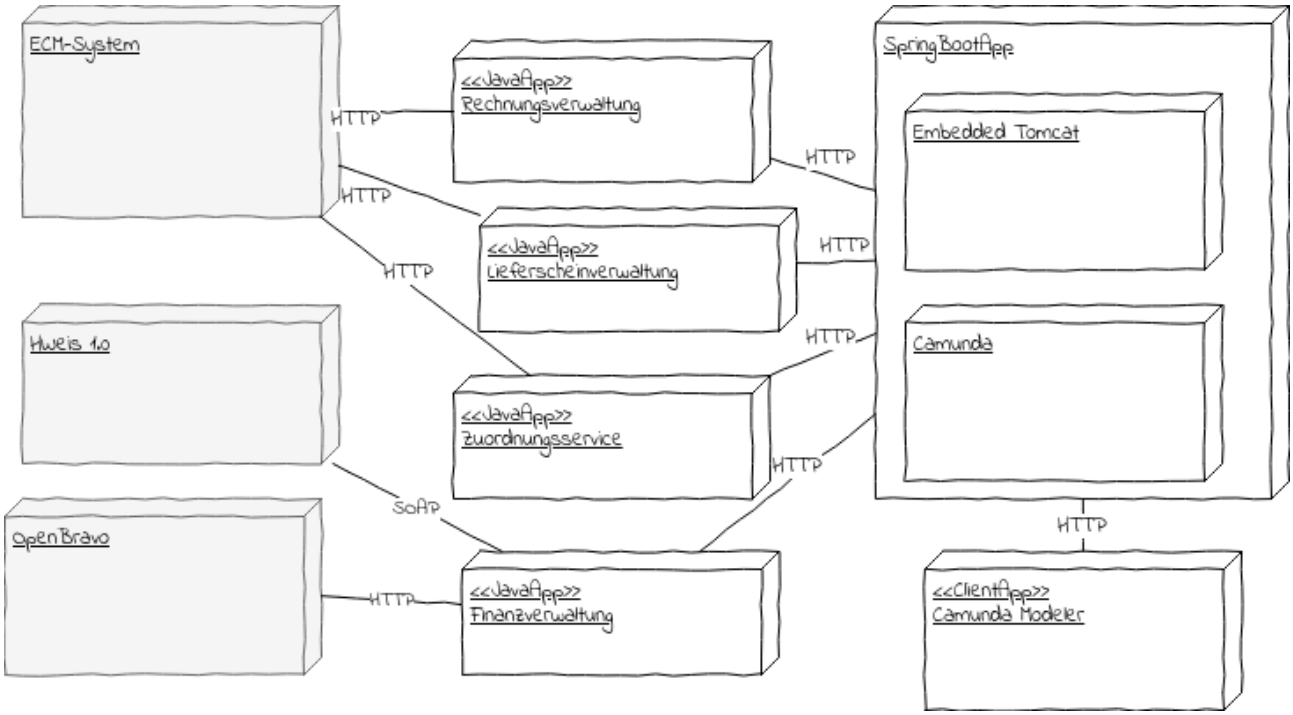

Abbildung 9: Verteilungssicht

Querschnittliche Konzepte

Domänenmodell

In Abbildung 10 ist das Domänenmodell dargestellt.

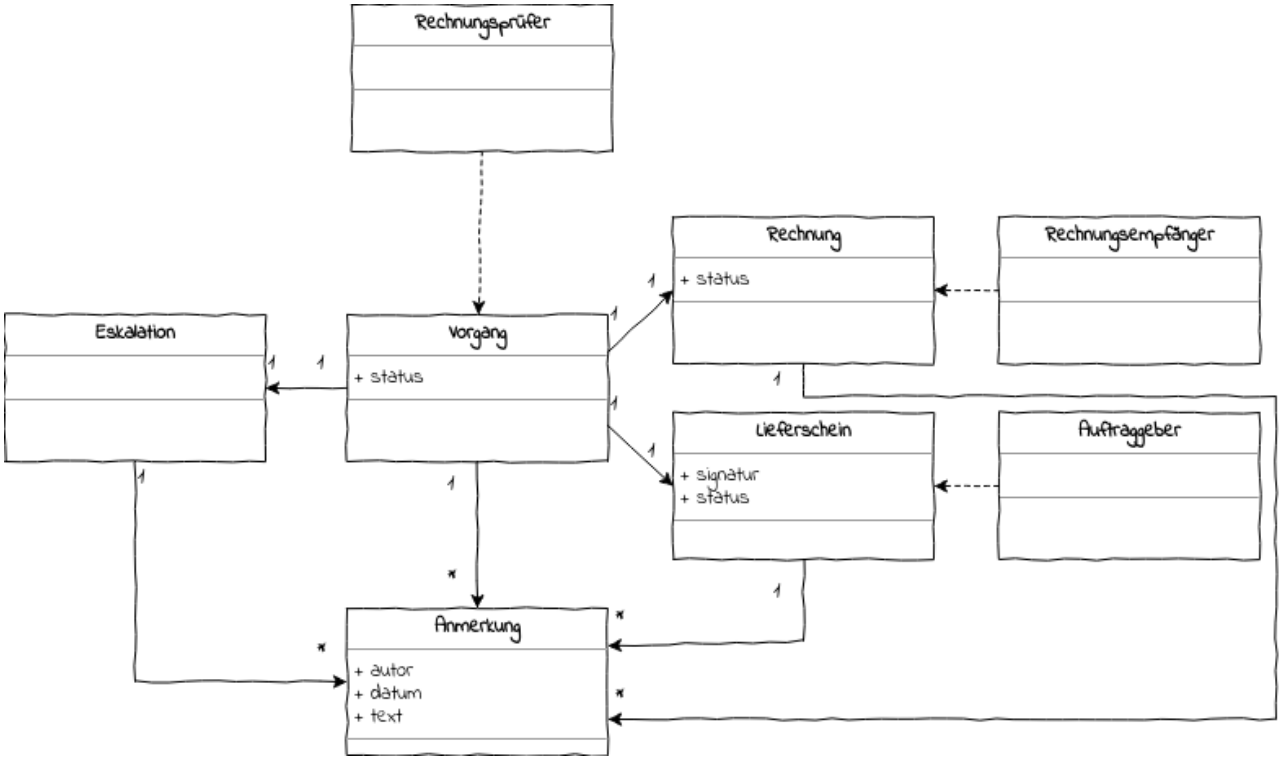

Abbildung 10: Geschäftsobjekte

Skalierbarkeit

Prinzip 1 - Klonen eines Services Camunda bietet zwei varianten Service Tasks auszuführen, entweder als *internal Service-Task* oder als *external Serivce-Task*. Bei einem internal Service-Task wird eine Java-Klasse, die das Camunda Interface JavaDelegation implementiert, ein Objekt, eine Objekt-Methode oder ein Skript aufgerufen. Die Funktionalität wird mit der Applikation deployed. Anders ist es bei external Service-Tasks. Diese Tasks werden nicht aufgerufen, sondern holen sich eigenständig ihre Aufgaben über HTTP von der Applikation (https://docs.camunda.org/manual/latest/user-guide/process-engine/external-tasks/). Bearbeitet ein external Service-Tasks eine Aufgabe, kann er die Aufgabe für andere Tasks sperren. Das erlaubt, dass jedem external Service-Tasks eine freie Aufgabe zugewiesen bekommt und das Skalieren, durch mehrfaches starten einer external Service-Tasks-Instanz bzw. Klonen, leicht möglich ist.

Prinzip 2 - Fachliche Aufteilung eines Services Es wurden die Fachlichkeiten „Rechnung in System einpflegen“ bzw. „Rechnung versenden“ getrennt. Das Gleiche gilt für die Digitalisierung des Lieferscheins.

Prinzip 3 - Teilung eines Service nach Consumer-Typen Die Services wurden nach Auftraggeber, Rechnungsempfänger, Rechnungsprüfer und Einkäufer getrennt.

Prinzip 4 - Teilung nach Read/Write Das Lesen bzw. Finden von zusammengehörigen Rechnungen und Lieferscheinen und das Schreiben bzw. Speichern der Zuordnung wurde getrennt.

Prinzip 5 - CQRS Die Umsetzung würde die Gesamtarchitektur komplexer machen. Ein Qualitätsziel des Prototypen war Verständlichkeit.

Prinzip 6 - Lokalisierung eines Service Services können an einem beliebigen Ort deployed werden. Die Kommunikation wird über HTTP vollzogen.

Prinzip 7 - Bildung von Caches Bisher werden keine Daten in eine Datenbank geschrieben, daher sind alle Daten nur lokal gespeichert bzw. „gecached“.

Kompensation

- Rechnung falschem Lieferschein zugeordnet → Zuordnung rückgängig machen
- Lieferschein kann keiner Rechnung zugeordnet werden → Zuordnung rückgängig machen
- Fehler bei der Transaktion → Rückbuchung veranlassen

Entwurfsentscheidungen

Reports

Im Prototyp werden keine Reports erstellt. Camunda enthält in der Enterprise-Edition Funktionen zum Erstellen von Reports. Alternativ könnten die Funktion durch Plug-ins oder zusätzliche Service Tasks realisiert werden.

Deployment

Camunda kann innerhalb von Spring Boot bzw. innerhalb des Spring Frameworks, über Docker oder „standalon“ z.B. in einem Tomcat-Server deployed werden. Für den Prototypen wurde entschieden, Camunda innerhalb einer Spring Boot-Anwendung zu deployen, da das Vorgehen neben Java keine Anforderungen an den Entwicklerrechner stellt. Für den Produktiveinsatz scheint die Verwendung von Docker die meiste Flexibilität (Verwendung eines Cloud-Dienstes oder On-Premise) zu bieten.

Cloud vs. On-Premise

Der Business-Case in Abbildung 11 zeigt, dass langfristig eine On-Premise Lösung zu bevorzugen ist.

On-Premise

Einsparung pro Quartal	
#Mitarbeiter	5
#Vorgänge/Mitarbeiter	200
Einsparung/Vorgang	5,00 €
Zwischensumme	5.000,00 €

Kosten	
Anschaffungskosten	48.000,00 €

Kosten pro Quartal	
Cloud-Anbieter	0,00 €

Jahr	2018				2019				2020			
Quartal	1	2	3	4	1	2	3	4	1	2	3	4
Einsparung	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €
Kosten	48.000,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €
Differenz	-43.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €
	-43.000,00 €	-38.000,00 €	-33.000,00 €	-28.000,00 €	-23.000,00 €	-18.000,00 €	-13.000,00 €	-8.000,00 €	-3.000,00 €	2.000,00 €	7.000,00 €	12.000,00 €
										ROI		

Kapitalwert	12.000,00 €

Coud

Einsparung pro Quartal	
#Mitarbeiter	5
#Vorgänge/Mitarbeiter	200
Einsparung/Vorgang	5,00 €
Zwischensumme	5.000,00 €

Kosten	
Anschaffungskosten	- €

Kosten pro Quartal	
Cloud-Anbieter	4.000,00 €

Jahr	2018				2019				2020			
Quartal	1	2	3	4	1	2	3	4	1	2	3	4
Einsparung	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €	5.000,00 €
Kosten	4.000,00 €	4.000,00 €	4.000,00 €	4.000,00 €	4.000,00 €	4.000,00 €	4.000,00 €	4.000,00 €	4.000,00 €	4.000,00 €	4.000,00 €	4.000,00 €
Differenz	1.000,00 €	1.000,00 €	1.000,00 €	1.000,00 €	1.000,00 €	1.000,00 €	1.000,00 €	1.000,00 €	1.000,00 €	1.000,00 €	1.000,00 €	1.000,00 €
	1.000,00 €	2.000,00 €	3.000,00 €	4.000,00 €	5.000,00 €	6.000,00 €	7.000,00 €	8.000,00 €	9.000,00 €	10.000,00 €	11.000,00 €	12.000,00 €

Kapitalwert	12.000,00 €

Abbildung 11: Business Case

Benutzereingabe

Camunda bietet verschiedene Möglichkeiten, Formulare zu erstellen:

Embedded Task Forms HTML-Dokument definieren, welches innerhalb der Tasklist angezeigt wird.

Generated Task Forms Formular innerhalb des Modellers mittels XML beschreiben.

External Task Forms HTML-Dokument definieren, auf welches verlinkt wird.

Generic Tasks Forms Der Anwender gibt Variablenname, Typ und Wert beim Starten eines Tasks an.

Für den Prototypen wurden Generated Task Forms verwendet, die Variante hat das beste Verhältnis zwischen Aufwand und Komfort geboten, um mit den BPM-Modell zu experimentieren.

Risiken und technische Schulden

Das System ist aktuell stark von der Camunda-Plattform abhängig. Da die Camunda-Plattform bereits in verschiedenen Unternehmen erfolgreich produktiv eingesetzt wird, kann davon ausgegangen werden, dass die Plattform stabiler läuft als es eine Eigenentwicklung zu beginn tun würde. Dennoch stellt die Camunda-Plattform eine Möglichkeit für einen Single Point of Failure dar. Einzelne Komponenten, wie beispielsweise die Taskslist oder das Cockpit könnten ggf. nach und nach durch Eigenentwicklungen ausgetauscht werden.

Es werden zurzeit keine Reports erstellt. Die Enterprise-Edition von Camunda bietet die entsprechende Funktion, die Anschaffung könnte jedoch über dem Budget des Unternehmens liegen.

Die Taskslist entspricht nicht dem Corporate Design des Unternehmens. Außerdem wurde für den Prototypen von Camunda generierte Formulare verwendet. Für komplexe Eingaben ist evtl. die Verwendung eines anderen Formulartypen vorteilhafter. Die Taskslist und die Formulare könnten zum Beispiel mit Angular, React oder Vue.js erstellt werden, siehe den Eintrag im Camunda-Blog (https://blog.camunda.com/post/2018/02/custom-tasklist-examples/).

BPM-Diagramme können schnell unübersichtlich werden, sobald neben dem „Happy-Path“ auch alle Ausnahmefälle modelliert werden. Dann ist vielleicht eine Aufteilung der Workflows auf verschiedenen Dateien nötig. Die Verwendung von zusätzlichen Elementen, wie beispielsweise Subprozessen, könnte ebenfalls helfen, die Übersicht zu verbessern.