ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Konzeption und Entwicklung einer Webservice Schnittstelle für ein Prozessmanagementsystem

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Andreas Alzner
andreas.alzner@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Klaus Kammerer

2016

Fassung 10. November 2016

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

Ziel von Unternehmen ist es die Effektivität und Effizienz ihrer Geschäftsprozesse zu steigern und somit Verwaltungskosten zu senken. Deshalb ist die Betrachtung dieser Geschäftsprozesse und deren Strukturierung, Analyse und Optimierung ein wichtiges Ziel. Diese Optimierungen können mithilfe von Prozessmanagement durchgeführt und unterstützt werden. Prozessmanagementsysteme, die den Lebenszyklus von Geschäftsprozessen unterstützen und eine automatisierte Ausführung von Tätigkeiten ermöglichen, sind hierbei eine wichtige Technologie. Damit eine Integration dieser Systeme und die Kommunikation zwischen vorhanden Informationssystemen und denselben implementiert werden kann, muss eine geeignete Schnittstelle vorhanden sein.

In dieser Arbeit wird ein Konzept einer generischen Webservice Schnittstelle zur Kommunikation zwischen beteiligten Informationssystemen und Prozessmanagementsystemen erstellt. Dabei bietet die Webservice Schnittstelle eine standardisierte Vorgehensweise zur Kommunikation in Verbindung mit einer einheitlichen Repräsentation der zu übermittelnden Daten an. Des Weiteren wird ein Mechanismus vorgestellt, mit dem die jeweiligen Informationssysteme ohne Schnittstellenveränderung ausgetauscht werden können. Zudem wurde eine prototypische Implementierung der Webservice Schnittstelle sowie ein Java Client für diese entwickelt.

Inhaltsverzeichnis

1 Einleitung

Jedes Unternehmen strebt eine Effizienz- und Effektivitätssteigerung seiner Geschäftsprozesse an, um Kosten zu reduzieren und beispielsweise Produkte schneller entwickeln und vertreiben zu können. Deshalb ist die Betrachtung von Geschäftsprozessen, welche Tätigkeiten in einem Unternehmen zusammenfassen, strukturieren und beschreiben, sowie die Analyse und Optimierung ein wichtiger Teil der Effizienz- und Effektivitätssteigerung und kann mithilfe von Prozessmanagement erreicht werden. Beim Prozessmanagement werden Geschäftsprozesse beispielsweise durch Interviews und Workshops erhoben, analysiert und optimiert. Dabei steht die Erreichung von definierten Geschäftszielen im Unternehmen stets im Vordergrund. Hierbei bietet sich die Einführung eines Prozessmanagementsystems an, welches das Management von Geschäftsprozessen, eine automatische Ausführung bestimmter Tätigkeiten in Geschäftsprozessen sowie eine systemgestützte Ausführung der Tätigkeiten durch Mitarbeiter eines Unternehmens ermöglicht. Die Integration eines Prozessmanagementsystems kann in jedem Unternehmen durchgeführt werden, beispielsweise in Krankenhäusern, Schulen und Universitäten. Häufig werden verschiedene Informationssysteme und Anwendungen gemeinsam mit einem Prozessmanagementsystem betrieben, deshalb ist die Kommunikation zwischen Informationssystemen und den einzelnen Anwendungen ein wichtiger Aspekt der Steigerung der Effizienz und Effektivität in einem Unternehmen.

1.1 Problemstellung

Viele Prozessmanagementsysteme bieten eine grafische Oberfläche zur Erstellung, Bearbeitung und Ausführung von Geschäftsprozessen an. Dennoch ist es in Unternehmen

notwendig angepasste Oberflächen und Anwendungen für deren Ausführung bereit zu stellen, die einfacher für Mitarbeiter zu bedienen sind und für bestimmte Aufgaben angepasst werden. Neben den Mitarbeitern sind auch Anwendungen und Schnittstellen für externe Prozessbeteiligte, beispielsweise Kunden oder Lieferanten, sinnvoll. Ein Beispiel hierfür ist die Bestellung in einem Online Versandhaus: hier initiiert der Kunde einen Bestellprozess, welcher in einem Prozessmanagementsystem abgebildet werden kann. Da viele Anwendungen in einem Unternehmen mit einem Informationssystem kommunizieren und dieses Informationssystem anschließend mit dem Prozessmanagementsystem arbeitet ist eine Kommunikation zwischen Informationssystemen und Prozessmanagementsystemen notwendig. Durch die Einbindung externe Prozessbeteiligter bietet sich eine generische Webservice Schnittstelle an. Diese kann die Kommunikation zwischen externen und internen Informationssystemen mit Hilfe eines Prozessmanagementsystems übernehmen. Da oftmals viele unterschiedliche Informationssysteme in einen Unternehmen eingesetzt werden, ist eine einheitliche Schnittstelle zwischen diesen notwendig. Hierdurch können Anwendungen ohne Kenntnis über die verwendeten Prozessmanagementsysteme beispielsweise über eine von vielen Programmiersprachen und Systemen unterstützte Webservice Schnittstelle kommunizieren.

1.2 Zielsetzung

Diese Arbeit stellt eine generische Webservice Schnittstelle vor, welche die Kommunikation zwischen beteiligten Informationssystemen und Prozessmanagementsystemen übernimmt. Dabei bietet die Webservice Schnittstelle eine einheitliche Schnittstelle und eine standardisierte Menge von Daten an. Die Webservice Schnittstelle bietet zusätzlich einen Mechanismus, der Informationen zur Mächtigkeit des Funktionsumfangs eines Prozessmanagementsystems und dessen Steuerung anbietet. Abschließend wird eine Beispielimplementierung anhand eines bestehenden Prozessmanagementsystems beschrieben.

1.3 Struktur der Arbeit

Nachdem dieses Kapitel sich mit der Motivation, der Problemstellung und der Zielsetzung befasst hat, werden im nachfolgenden Kapitel 2 Grundlagen beschrieben, die zur Entwicklung von Webservice Schnittstellen benötigt werden.

Kapitel 4 beschreibt die Prozessmanagementsysteme *Clavii BPM Platform* und *Activiti BPM Platform* und deren Architektur. Die Webservice Schnittstelle wurde prototypisch auf Basis der Clavii BPM Platform entwickelt, deshalb ist die Betrachtung dessen Architektur notwendig.

Kapitel 3 definiert Anforderungen einer Webservice Schnittstelle, anschließend wird ein geeignetes Architekturschema zur prototypischen Umsetzung vorgestellt. Zusätzlich werden in diesem Kapitel die Anforderungen des zu verwendenden Webservice Protokolls beschrieben und ein Schnittstellenmodell skizziert. Die Realisierung der in Kapitel 3 vorgestellten Architektur wird in Kapitel 5 näher beschrieben und mit dem Konzept des Schnittstellenmodells verglichen. Neben der Realisierung der Webservice Schnittstelle wird in Kapitel 3 eine Java Client API entwickelt, die eine einfache Verwendung der erstellen Webservice Schnittstelle erlaubt.

Kapitel 6 stellt zwei Prozessmanagementsysteme mit Webservice Schnittstellen vor und vergleicht diese mit der im Rahmen dieser Arbeit entwickelten Webservice Schnittstelle. Abschließend wird in Kapitel 7 die Arbeit zusammengefasst und ein Ausblick über zukünftige Entwicklungsschritte skizziert.

2

Grundlagen

2.1 Geschäftsprozessmanagement

Geschäftsprozessmanagement (engl.: *Business Process Management (BPM)* bezeichnet den Vorgang der Erhebung, Analyse, Optimierung, Ausführung und Überwachung von Geschäftsprozessen in einem Unternehmen. In einem Geschäftsprozess werden Tätigkeiten und Strukturen festgelegt und dokumentiert, die in einem Unternehmen bei dessen Abläufen zur Wertschöpfung, sowie unterstützenden Abläufen anfallen. Dabei werden diese Geschäftsprozesse mithilfe von Vorgehensweisen und Techniken im Rahmen von Prozessmanagement erhoben, näher spezifiziert und durch Interviews und Workshops analysiert und optimiert. Da Geschäftsprozesse anschließend verbessert werden, können Steigerungen in der Effizienz und Effektivität erreicht werden, da beispielsweise die Ausführung von festgelegten Tätigkeiten schneller und strukturierter durchgeführt werden kann. Deshalb können auch Produktionskosten gesenkt oder Dienstleistungen schneller durchgeführt werden, die wiederum zu einer höheren Kundenzufriedenheit oder einer besseren Produktqualität führen können [1]. Ein Beispiel für einen Geschäftsprozess ist die Abwicklung einer Bestellung in einem Online-Warenhaus (siehe Abbildung 2.1).

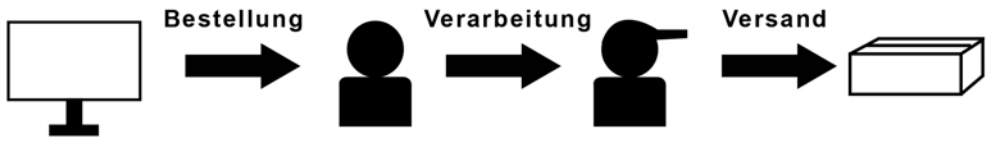

Abbildung 2.1: Vereinfachter Bestellprozess

In diesem vereinfachten Beispiel (siehe Abbildung 2.1) wird eine Online-Bestellung eines Kunden dargestellt. Hierbei wird eine Online-Bestellung von einem Mitarbeiter des Unternehmens bearbeitet und danach durch einen Lagerarbeiter versandbereit vorbereitet. Abschließend wird das Produkt an den Kunden versendet. In diesem Beispiel wurde die Überprüfung des Lagerbestandes, als auch die Abwicklung der Bezahlung nicht betrachtet. Es handelt sich um einen Geschäftsprozess, der noch nicht genauer dokumentiert und optimiert wurde.

Damit ein Geschäftsprozess nach der Gestaltung und Implementierung optimiert werden kann, muss dieser in regelmäßigen Abständen evaluiert, analysiert und verbessert werden. Dadurch empfiehlt sich die genauere Betrachtung des *Prozesslebenszyklus* (engl.: *business process lifecycle*), der im nachfolgenden Abschnitt erläutert wird.

2.1.1 Einführung

Ein wichtiger Bestandteil des Prozessmanagements ist der Prozesslebenszyklus (siehe Abbildung 2.2). Dieser Lebenszyklus veranschaulicht die einzelnen Phasen, die Geschäftsprozesse und deren Prozessmodell durchlaufen. Ein Prozessmodell ist eine Spezifikation von Geschäftsprozessen in einer bestimmten Notation (z.B. BPMN 2.0). Die Phasen des Prozesslebenszyklus sind in einer zyklischen Struktur angeordnet, um deren logische Abhängigkeit untereinander darzustellen [1].

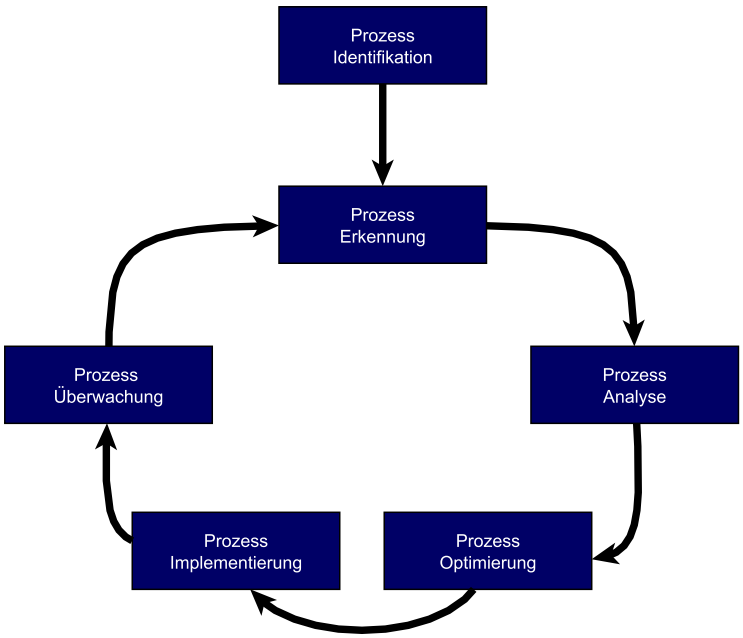

Abbildung 2.2: Geschäftsprozesslebenszyklus (nach [1])

Der Prozesslebenszyklus beschreibt die Phasen *Prozessidentifikation*, *Prozesserhebung*, *Prozessanalyse*, *Prozessoptimierung*, *Prozessimplementierung* und *Prozessüberwachung* (siehe Abbildung 2.2).[1].

Die Prozessidentifikationsphase ist außerhalb des Lebenszyklus angeordnet und ist der Einstiegspunkt für die Erstellung eines Prozessmodells. In dieser Phase werden Geschäftsprozesse identifiziert. Dadurch entsteht eine Sammlung an Geschäftsprozessen und deren Relationen zueinander, die näher analysiert, optimiert und technisch in einem Prozessmanagementsystem implementiert werden können. In der Prozesserhebungsphase werden die einzelnen Geschäftsprozesse genauer betrachtet und mithilfe von Interviews, Workshops oder Prozess Mining genauer beschrieben und dokumentiert. Dadurch wird eine Sammlung von Prozessmodellen und den zugehörigen Beschreibungen erstellt. Die Phase der Prozessanalyse beinhaltet die Analyse der Geschäftsprozesse zur Erkennung von Problemen und Schwachstellen, als auch die Betrachtung von Auswirkungen auf die Performanz und Effizienz der einzelnen Geschäftsmodelle in einem Unternehmen. Diese Phase ist ein wichtiger Bestandteil

zur Verbesserung von Prozessmodellen, da existierende Probleme oder Schwachstellen, die Organisationsstruktur, verfügbare Ressourcen und involvierende IT-Systeme beeinträchtigen könnten. Als Ergebnis dieser Phase erhält man eine Sammlung an Problemen und Schwachstellen. Anschließend werden die zuvor analysierten und erkannten Probleme in der Phase der Prozessoptimierung verbessert. Dadurch entsteht eine optimierte Version eines Geschäftsprozesses, der für eine technische Implementierung in einem Prozessmanagementsystem geeignet ist. Die Implementierung der einzelnen Geschäftsprozesse werden in dieser Phase durchgeführt. Dabei müssen die Geschäftsprozesse auf ein bestimmtes Prozessmanagementsystem angepasst werden und in die bestehende organisatorische Struktur des Unternehmens integriert werden. Hierbei kann es vorkommen, dass Änderungen der zuvor erstellten Prozessmodellstruktur notwendig sind. Das Ergebnis ist ein automatisierter Geschäftsprozess, der ausgeführt und von einzelnen Prozessbeteiligten verwendet werden kann. Nachdem die Konfigurierung und Implementierungsphase beendet wurde, kann ein Prozessmodell ausgeführt werden. Die Phase der Ausführung umfasst die aktive Laufzeit eines Geschäftsprozesses. Dabei kontrolliert ein Prozessmanagementsystem die Ausführung der einzelnen Geschäftsprozessinstanzen und deren Laufzeitverhalten, wie zuvor im Prozessmodell definiert wurde. Eine Monitoringkomponente des Prozessmanagementsystems visualisiert hierbei den Status der Geschäftsprozessinstanzen. Die Phase der Prozessüberwachung ist ein wichtiger Mechanismus, um den genauen Zustand und die Informationen einer Instanz anzuzeigen. Durch Monitoring lassen sich auch weitere Informationen ableiten, wie beispielsweise die durchschnittliche Ausführungsdauer einer Prozessinstanz, Probleme bei der Ausführung, usw. Während der Ausführung werden die erzeugten Daten in Log-Dateien gespeichert. Diese Log-Dateien bestehen aus geordneten Mengen von Ereignissen, die während der Ausführung der einzelnen Prozessinstanzen erzeugt werden [1].

Am Lebenszyklus eines Geschäftsprozesses sind verschiedene Personen beteiligt, die nachfolgend kategorisiert werden:

- *Chief Process Officer (Prozessmanagement Leiter):* Der Chief Process Officer ist für die Standardisierung und der Harmonisierung der Geschäftsmodelle und Geschäftsprozesse zuständig. Zusätzlich ist er dafür verantwortlich, Geschäftspro-

zesse zu verbessern und anzupassen, z.B. um auf Marktveränderungen reagieren zu können.

- *Business Engineer:* Business Engineer sind unternehmensbezogene Experten, die für das Definieren von strategischen Zielen des Unternehmens verantwortlich sind. Meistens besitzen Business Engineers kaum IT-Kenntnisse, weswegen eine einfach zu verwendende Notation zur Modellierung verwendet werden sollte.

- *Process Designer (Prozess Gestalter):* Process Designer sind für die Modellierung der Geschäftsprozesse verantwortlich. Sie entwickeln durch Kommunikation mit Business Engineers und anderen Personen Prozessmodelle.

- *Process Participant (Prozessbeteiligter):* Ein Process Participant ist eine Person, die an der Ausführung von Geschäftsprozess-Tätigkeiten beteiligt sind. Ihre Meinungen und Erfahrungen sind während der Entwicklung der Geschäftsmodelle besonders wichtig, da Sie Ihre Kenntnisse für eine Verbesserung der einzelnen Geschäftsprozesse einbringen können.

- *Knowledge Worker:* Ein Knowledge Worker ist ein Prozessbeteiligter, der direkt die Aktivitäten in einem Informationssystem ausführt. Seine Kenntnisse sind ebenfalls wichtig für die Entwicklung und Optimierung einzelner Aktivitäten innerhalb eines Geschäftsprozesses.

- *Process Responsible (Prozessverantwortlicher):* Jeder Geschäftsprozess besitzt eine bestimmte Person, die für sie verantwortlich ist. Diese ist für die korrekte und effiziente Ausführung der Geschäftsprozesse verantwortlich. Zusätzlich ist diese Person dafür verantwortlich, Ineffizienz und Probleme zu erkennen und diese gemeinsam mit den Process Designern und den Process Participants zu verbessern.

- *System Architekt:* Sie sind für die Implementierung und Konfiguration der Prozessmanagementsysteme verantwortlich. Ihre Aufgabe besteht darin, die Prozessmodelle so zu konfigurieren, dass diese auf einem Prozessmanagementsystem ausgeführt werden können.

- *Developer (Entwickler):* Developer entwickeln Anwendungskomponenten, auf die beispielsweise während der Ausführung einer Prozessinstanz zurückgegriffen werden kann. So entwickelt eine Developer beispielsweise eine Anwendungskomponente, die einen automatischen Versand von E-Mails ermöglicht.

Es existieren verschiedene Notationen, um Geschäftsprozesse in Prozessmodellen darzustellen. Alle Graphiken und Beispiele die in dieser Arbeit verwendet wurden basieren auf der grafischen Beschreibungssprache *BPMN 2.0*. Im Folgenden wird genauer auf diese Notation eingegangen.

2.1.2 BPMN 2.0

BPMN 2.0 ermöglicht eine grafische Modellierung von Geschäftsprozessen in Prozessmodellen. Ein Prozessmodell ist hierbei eine zeitlich-sachlogische Abfolge von Aktivitäten oder Tätigkeiten. Zusätzlich lassen sich BPMN-Prozessmodelle mit anderen Beschreibungssprachen kombinieren und erweitern. Wichtig bei der Entwicklung von BPMN-Prozessmodellen ist die Frage, welche Personen an einem Geschäftsprozess beteiligt sind.

Im Folgenden werden die einzelnen Prozesselemente von BPMN 2.0 eingeführt und anhand von Prozessmodellbeispielen erläutert.

BPMN Kernelemente

Die BPMN umfasst eine große Menge an grafischen Prozesselementen, mit denen ein Prozessmodell erstellt werden kann. Für einige Prozesselemente existieren verschiedene Typen und Varianten, mit denen ein Prozessmodell genauer modelliert werden kann.

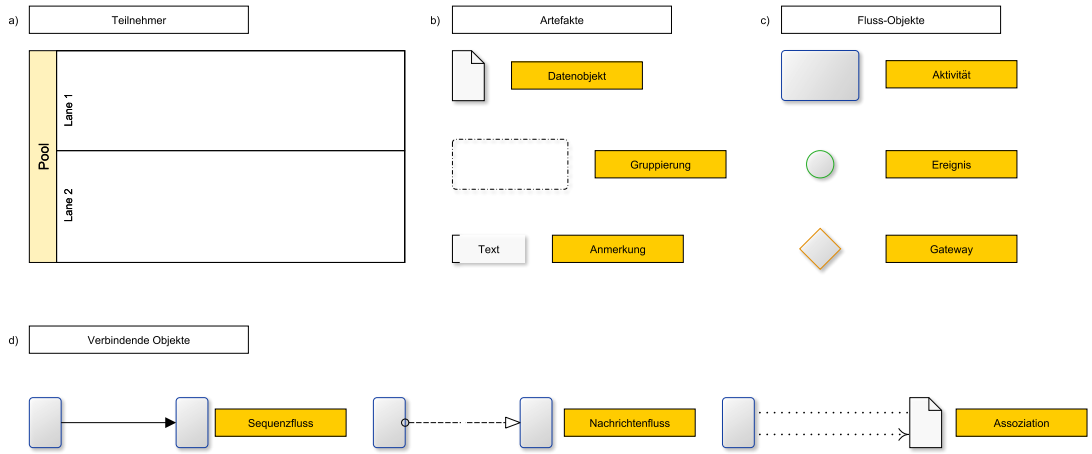

Abbildung 2.3: BPMN Kernelemente im Überblick (nach [2])

Bei der Modellierung mit BPMN 2.0 werden die Kernelemente verwendet und zu einem Prozessmodell zusammengefasst. Hierbei lassen sich die Kernelemente in den Gruppen Teilnehmer, Artefakte, Fluss-Objekte und den verbindenden Objekten zusammenfassen (siehe Abbildung 2.3) [2]. Dabei bilden die Fluss-Objekte die wichtigste Grundlage zur Modellierung, da diese Elemente die Aktivitäten und Ereignisse in einem Prozessmodell beschreiben. Die Fluss-Objekte, die in der Abbildung 2.3c abgebildet wurden, beinhaltet die Kernelemente Aktivität, Ereignis und Gateway. Aktivitäten beschreiben Elemente, die in einem Prozessmodell ausgeführt werden müssen. Dies sind die eigentlichen Aktivitäten in einem Prozessmodell. Unter Umständen werden Aktivitäten nur unter bestimmten Bedingungen ausgeführt. Um Bedingungen zu beschreiben und darzustellen werden Gateways verwendet. Ereignisse beschreiben ein bestimmtes Geschehen im Prozessmodell, wie beispielsweise der Start des Prozesses oder das Ende eines Prozesses. Damit der Ablauf eines Prozessmodelles beschrieben werden kann werden Elemente der verbindenden Objekte verwendet. Unter verbindenden Objekten werden Sequenzfluss, Nachrichtenfluss und Assoziation gruppiert. Flussobjekte werden innerhalb eines Pools (bzw. einer Lane) durch Sequenzflüsse verbunden. Falls eine Verbindung über Poolgrenzen hinweg erfolgt werden Nachrichtenflüsse verwendet. Artefakte, die eigentlich keinen Einfluss auf die Reihenfolge der Flussobjekte haben, werden mit Assoziationen an Fluss-Objekten gebunden. Beispiele wurden in Abbildung 2.3d abgebildet. Dabei können

bestimmte Aktivitäten und Ereignisse durch verschiedene Teilnehmer ausgelöst oder ausgeführt werden. Eine *Lane* representiert einen Teilnehmer in einem Prozessmodell. Ein Beispiel für einen *Pool* und einer Lane ist in Abbildung 2.3a. Ein Teilnehmer kann eine bestimmte Rolle oder ein bestimmter Mitarbeiter sein. Diese Teilnehmer werden durch einen Pool gruppiert und zusammengefasst. Beispielsweise können in einem Pool mit den Namen ’Enterprise’ die Lanes ’Buchhaltung’, ’Lager’ und ’Geschäftsführung’ vorhanden sein. Unter den Artefakten (siehe Abbildung 2.3b) befinden sich Elemente wie Datenobjekte, Gruppierung und Anmerkung. Datenobjekte werden verwendet um bestimmte Datenflüsse zu kennzeichnen. Mit Gruppierungen lässt sich eine Anzahl von Elementen zu einer Gruppe zusammenfassen und durch die Anmerkung lassen sich Textuelle Anmerkungen verfassen (Bps. später für komplexe Gateways verwendet).
Als einfaches Beispiel wird die Abbildung 2.1 als Prozessmodell modelliert. Hierbei werden nur Blanko-Ereignisse verwendet (Abb.2.4).

Abbildung 2.4: Prozessmodell einer Bestellung mit Blanko-Ereignissen [2]

In unserem Modell (siehe Abbildung 2.4) wird der Prozess durch die Bestellung eines Artikels eingeleitet. Nach der Bestellung des Artikels wird ein Blanko-Zwischenereignis an die Buchhaltung gesendet. Die Buchhaltung bucht die Bestellung und leitet den Versand im Lager ein. Anschließend verarbeitet das Lager die Bestellung und versendet diese. Dieser Prozess endet mit der Annahme des versendeten Pakets. Das Modell beschreibt die einfachste Darstellung einer Bestellung, dabei wurden einige Aktivitäten (Bezahlung, Artikel verpacken usw.) wegen der einfachen Lesbarkeit weggelassen.

Damit eine große Variation an Prozessmodellen mit individuellen Abläufen modellierbar ist, werden verschiedene Gateways und Ereignisse in BPMN 2.0 definiert. Diese werden in den nachfolgenden Kapitel 2.1.2 und 2.1.2 aufgelistet und beschrieben.

Gateways

In BPMN gibt es eine vielzahl an verschiedenen Gateways. Es werden nur die am häufigsten verwendeten Gateways aufgelistet (Abb.2.5) und beschrieben.

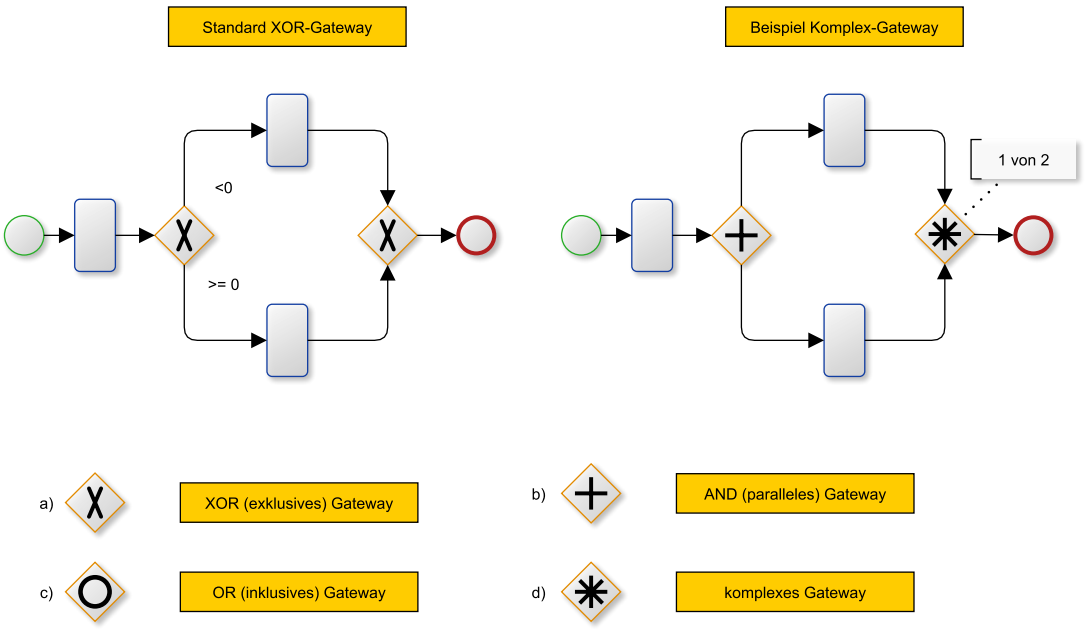

Abbildung 2.5: BPMN 2.0 Gateways

Beim exklusiven Gateway handelt sich um eine Bedingung bei der anschließend ein bestimmter Pfad gewählt wird. Ein exklusives Gateway ist in Abbildung 2.5a abgebildet. Hierbei wird immer nur ein Pfad ausgewählt. Die Bedingungen an denen die Pfade gerichtet sind sollten alle Möglichkeiten abdecken. Wie in der Abbildung 2.5 im Linken Standard XOR-Gateway 2 Pfade auswählbar sind. Wird die Entscheidung des Pfades anhand einer Zahl getroffen so müssen die Bedingungen der Pfade alle Zahlen abdecken. Beispielsweise könnte der erste Pfad alle Zahlen <0 und der zweite Pfad alle Zahlen >= 0 abdecken. Bei einem inklusiven Gateway können ein, mehrere oder keine Pfade ausgewählt werden. Werden die Bedingungen dieses Gateways automatisiert müssen nicht alle Möglichkeiten abgedeckt werden. Trifft keine Bedingung zu wird kein Pfad gewählt und die nächste Aktivität wird ausgeführt. Treffen mehrere Bedingungen

zu werden alle Pfade der zutreffenden Bedingungen angefahren. Dargestellt wird ein inklusives Gateway, wie in Abbildung 2.5c gezeigt, durch eine Route mit einem Kreis in der Mitte. Hierbei handelt es sich um ein Gateway bei denen alle Pfade angesteuert werden (siehe Abbildung 2.5b) und nur fortgesetzt wird wenn alle Pfade beendet worden sind. Dadurch können parallele Aktivitäten abgebildet werden. Das komplexe Gateway findet überall dort Anwendung in denen ein Verhalten gefordert wird, welches mit den anderen Gateways alleine nicht modelliert werden kann (siehe Abbildung 2.5d). Ein Beispiel hierbei wäre: Wir befragen 5 Gutachter zu einem Unfall und möchten unsere Entscheidung treffen sobald 3 Meinungen der Gutachter eingegangen sind. Das Problem ist, mit einem OR-Gateway oder einem AND-Gateway lässt sich dieses Verhalten nicht realisieren, da beide Gateways anschließend auf die Beendigung aller Pfade (aller 5 Gutachter) warten. Deswegen benötigen wir nun das komplexe Gateway, hier können wir als Bedingung hinzufügen 3 von 5, das Bedeutet bei 3 Meinungen wird der Prozess fortgeführt und alle weiteren Meinungen werden nicht mehr weiter betrachtet (Siehe Abb.2.5 Beispiel Komplexes-Gateway) [2].

Ereignisse

Ein Ereignis stellt immer ein Geschehen dar, welches vor, während oder am Ende eines Prozesse stattgefunden hat [2]. BPMN definiert viele verschiedene Ereignisse und im Beispiel (Abb.2.4) wurden Startereignis, Zwischenereignis und Endereignis verwendet. In Abbildung 2.6 findet sich eine Liste an ausgewählten Ereignissen.

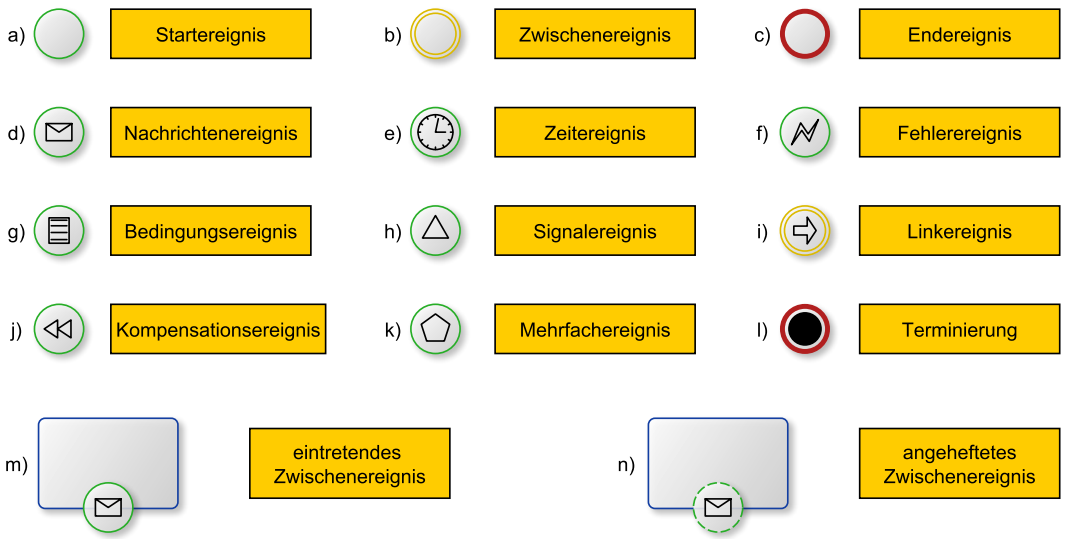

Abbildung 2.6: BPMN 2.0 Ereignisse

Das *Startereignis* (siehe Abbildung 2.6a) ist der Start eines Prozesses. Dieses Ereignis zeigt in der Regel den Grund an warum ein Prozess gestartet wurde. Ein *Zwischenereignis* (siehe Abbildung 2.6b) stellt immer den Status eines Prozesses dar, den man im Prozessmodell festhalten möchte. Jedoch werden Zwischenereignisse selten verwendet. Endereignisse (siehe Abbildung 2.6c) stellen das Ende eines Prozesspfades dar. Durch ein *Nachrichtenereignis* (siehe Abbildung 2.6d) lassen sich Kommunikationen in einem Prozess abbilden. Dabei beschränkt sich der Begriff Nachricht nicht nur auf Briefe, E-Mails oder Anrufe. Mit einem *Zeitereignis* (siehe Abbildung 2.6e) lassen sich verschiedene Zeitabhängige Ereignisse generieren. Ein Zeitereignis als Startereignis kann als Intervall (Bsp. alle 2 Stunden), Regelmäßigkeit (Bps. Mo-Fr um 8:00 Uhr) oder einmalig zu einem bestimmten Zeitpunkt (Bps. Am 22.12 um 10:00 Uhr) verwendet werden.
Zeitereignisse als Zwischenereignisse können als Timer zu einem bestimmten Zeitpunkt (Bsp. bis 18 Uhr), Timer zu einer Zeitspanne (Bsp. 14:30 Uhr bis 16 Uhr) oder zu einem Timer nach einer Aktivität (Bsp. nach 30 Minuten inaktivität) verwendet werden. Falls in einem Prozess Fehler (Exceptions) auftauchen oder geworfen werden, dann können Fehlerereignisse (siehe Abbildung 2.6f) in BPMN durch einen Blitz dargestellt werden.

Wie ein Fehler in BPMN genau definiert wird ist nicht genauer spezifiziert. Ein *Bedingungsereignis* (siehe Abbildung 2.6g) beschreibt ein Ereignis, bei dem ein Prozess nur gestartet oder fortgesetzt werden kann wenn eine bestimmte Bedingung erfüllt ist. Diese Bedingung kann unabhängig vom Prozess erfüllt werden. Beispiel: Bei unserem Prozess (Abb.2.4) kann ein Artikel nur versendet werden, falls alle Teile verfügbar sind. Somit könnte man eine Bedingung 'Alle Artikel verfügbar' einfügen, der nicht im eigentlichen Prozess erfüllt wird. Das *Signalereignis* (siehe Abbildung 2.6h) ähnelt dem Nachrichtenereignis sehr, allerdings besteht der wesentliche Unterschied darin, dass das Nachrichtenereignis an einen bestimmten Empfänger gerichtet ist, wohingegen das Signalereignis ungerichtet ist. Ein *Linkereignis* (siehe Abbildung 2.6i) ist ein Sonderfall der keiner speziellen Bedeutung zuzuordnen ist. Mit einem Linkereignis lassen sich Brücken in einer Prozessmodell bauen, falls sich Sequenzflüsse oder Nachrichtenflüsse schneiden würden. Bei einem *Kompensationsereignis* (siehe Abbildung 2.6j) geht es generell in Prozessen um Aufgaben die bei einer bestimmten Bedingung Rückgängig gemacht werden müssen. Beispiel: In Beispiel aus Abbildung 2.4 wird die Bestellung des Artikels während der Bearbeitung im Lager storniert. Nun kann durch die Verwendung des Kompensationsereignisses die Reservierung der Artikel rückgängig gemacht werden und die Bestellung storniert werden. Bei einem *Mehrfachereignis* (siehe Abbildung 2.6k) lassen sich mehrere Ereignisse gleichzeitig auslösen oder empfangen. Beispielsweise kann ein Mehrfachereignis sowohl das Ereignis eines Timers zum Abbruch als auch das Nachrichtenereignis zum abbrechen entgegennehmen und eine Abbruchaktion ausführen. Die Terminierung (siehe Abbildung 2.6l) findet seine Anwendung in Prozessen in denen mehrere Pfade gleichzeitig abgearbeitet werden, allerdings der Prozess bei Beendigung eines bestimmten Pfades oder unter einer bestimmten Bedingung beendet werden soll. Erreicht eine Prozessinstanz eine Terminierung so wird der ganze Prozess beendet. Tritt ein Zwischenereignis während der Ausführung dieser Aktivität (siehe Abbildung 2.6m) auf, so wird diese laufende Aktivität unterbrochen und die Abhandlung des Ereignisses gestartet. Dies betrifft die laufende Aktivität als auch all ihre nachfolgenden Aktivitäten, da die laufende Aktivität nicht beendet wird. Aktivitäten mit dieser Eigenschaft werden Aktivität mit eintretendes Zwischenereignis genannt. Oftmals kann das Verhalten des eintretenden Zwischenereignisses bei einer Aktivität, durch

die Beendigung der Aktivität, zu Problemen führen. Deswegen kann in BPMN 2.0 das Symbol des nicht-unterbrechende angeheftete Zwischenergebnis (siehe Abbildung 2.6n) verwendet werden. Hierbei wird die laufende Aktivität nicht unterbrochen sobald das Zwischenereignis eintritt und zeitgleich die Abhandlung des Zwischenereignisses parallel gestartet werden [2].

Teilprozesse

Ein Prozessmodell kann durch mehrere Teilprozesse repräsentiert werden und dadurch an Übersicht gewinnen. Ein weiterer Vorteil ist die Wiederverwendbarkeit der Teilprozesse in verschiedenen Prozessmodellen [2].
In der Abbildung 2.7 wurden bestimmte Aktivitäten zu Teilprozesse zusammengefasst. Diesmal wurde das Beispiel des einfachen Prozesses (siehe Abbildung 2.4) um einen Pool und den verschiedenen Teilnehmern Lieferant, Buchhaltung, Lager und Lieferant erweitert. Zusätzlich wird nun beim Versand der Bestellung aus dem Lager zum Lieferanten und vom Lieferanten zum Käufer Nachrichtenereignisse verwendet um den Versand des Paketes besser darzustellen.

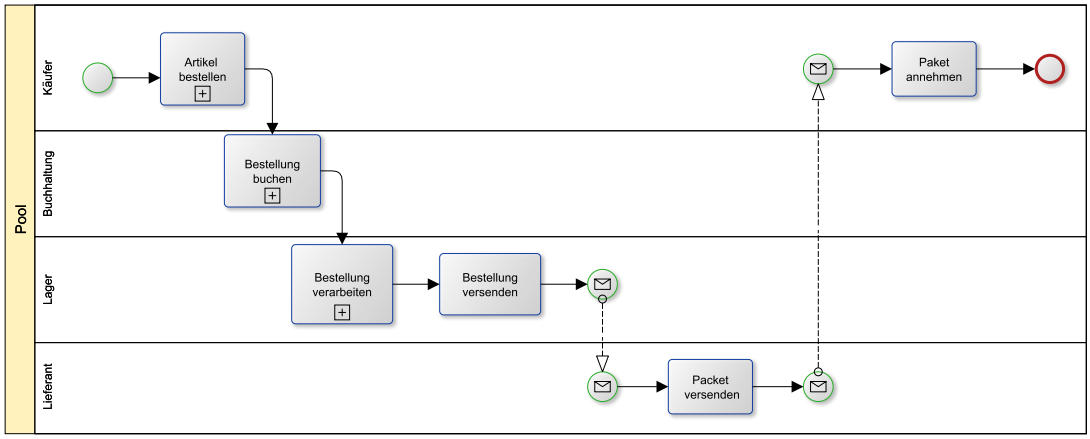

Abbildung 2.7: Bestellprozess mit Teilprozessen

Die Teilprozesse in Abbildung 2.7 fassen bestimmte Aktivitäten zusammen. Der Gesamtprozess ist in Abbildung 2.8 abgebildet.

- *Artikel bestellen:* Dieser Teilprozess fasst den Ablauf der Artikelbestellung zusammen. Hierbei kann der Käufer mehrere Artikel in den Warenkorb legen (Loop). Er kann sich bei jeder Iteration für 'weiter einkaufen', 'fertig' und 'abbrechen' entscheiden. Die Aktivitäten zur Bearbeitung des Warenkorbs und der Entfernung von Artikeln aus dem Warenkorb wurden hierbei vernachlässigt. Wählt der Käufer abbrechen aus, so wird der gesamte Prozess beendet. Ist der Käufer mit seinen Artikeln fertig so kann er den Warenkorb erneut betrachten. Anschließend wird der Kunde zur Kasse gebeten und die Bestellung abgeschlossen.
- *Bestellung buchen:* Dieser Teilprozess fasst die Aktivität Bestellung buchen zusammen. Zunächst verarbeitet die Buchhaltung die bestellten Artikel und erstellt die Rechnung parallel zur Reservierung der Artikel. Nachdem die Rechnung erstellt wurde wird die Rechnung zum Kunden Versand und auf die Zahlung des Kunden gewartet. Nachdem die Zahlung abgeschlossen wurde und die Artikel reserviert wurden wird der Versand der Artikel an das Lager weitergeleitet.
- *Bestellung verarbeiten:* In diesem Teilprozess wird die Lieferung vorbereitet, dabei werden parallel die Artikel verpackt und die Lieferung vorbereitet und der Liefertermin festgelegt. Nachdem kann die Lieferung des Artikels an den Kunden weitergeleitet werden.

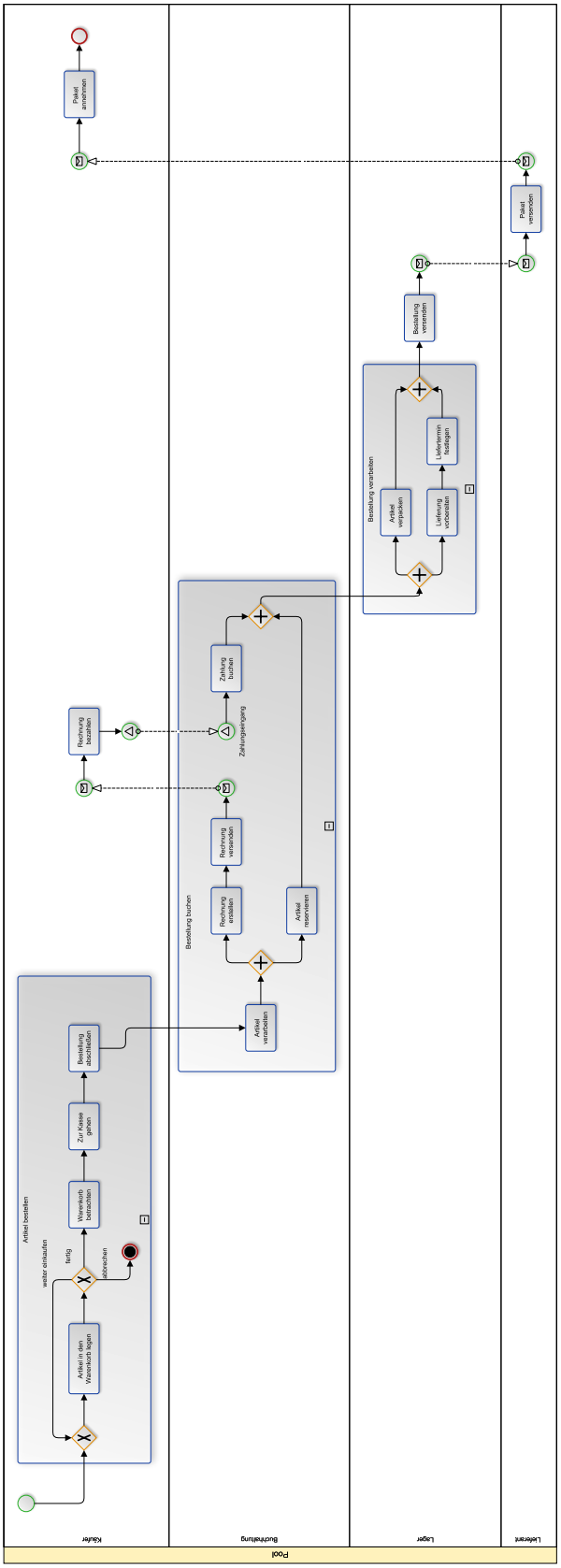

Abbildung 2.8: Bestellungprozess ohne Teilprozesse

2.1.3 Prozessmanagementsystem

Zwar ist das Interesse an Prozessmanagementsystemen gestiegen, dennoch arbeiten viele Unternehmen bislang noch manuell und mit handlich bearbeiteten Dokumenten, anstatt die Ausführung der Prozesse durch ein Prozessmanagementsystem durchzuführen [3]. Ein Prozessmanagementsystem unterstützt die Verwaltung und Ausführung von Geschäftsmodellen und bringt einige wesentliche Gründe für die Einführung von Prozessmanagementsystemen in einem Unternehmen:
Eine aktive Prozessunterstützung, die den Mitarbeitern oder Bearbeitern eine Aufgabenliste und ein Ablaufschema bereitstellt. Diese Aufgaben können zudem von den Mitarbeitern ausgeführt werden.
Teilaufgaben und Teilprozesse können automatisiert werden, dadurch müssen nicht alle Aufgaben manuell vollzogen werden. Ändern sich außerdem die Gegebenheiten oder der Markt eines Unternehmens können die Prozessmodelle schnell und flexibel an die neuen Abläufe angepasst werden. Zusätzlich lassen sich Prozesse besser überwachen und Fehler oder Probleme können leichter nachvollzogen werden. Zur Überwachungen stehen zudem noch Performance-Daten zur Verfügung, wie beispielsweise die durchschnittliche Ausführungszeit eines Prozesses.

Zusätzlich wird von einem Prozessmanagementsystem die Möglichkeit der beliebigen Verknüpfung von Anwendungsfunktionen (Services und Dienste) erwartet. Außerdem sollten Prozessmanagementsysteme Flexibilität zur Entwurfszeit, Flexibilität zur Ausführungszeit und Prozessschemaevolution bieten [3].

- *Flexibilität zur Entwurfszeit*
  Dies bedeutet, dass man rasch neue Prozessmodelle erstellen kann und diese Modelle nachbearbeitet werden können. Dies ist bei den meisten Prozessmanagementsystemen grafisch gelöst worden.

- *Flexibilität zur Ausführungszeit*
  Dabei geht es um die Bearbeitbarkeit auf Prozessinstanzebene. Hierbei kann das Modell während der Ausführung erweitert und bearbeitet werden. Dadurch können bestimmte Instanzen zur Laufzeit individualisiert werden.

- *Prozessschemaevolution*
  Dies ist die Fähigkeit eines Prozessmanagementsystems laufende Prozessinstanzen an ein geändertes Prozessmodell bzw. Schema zu migrieren. Dies ist nur möglich soweit eine Prozessinstanz nicht zu vorangeschritten ist.

Die in dieser Arbeit verwendete Clavii BPM Platform basiert auf Activiti und gehört zu den Prozessmanagementsystemen die ein Großteil der Anforderung von Prozessmanagementsystemen unterstützt.

2.2 Service-orientierte Architekturen (SOA)

Eine *Service-orientierte Architektur (SOA)* ist ein Muster zur Organisation und Zusammenfassung von Funktionen als Services. Diese Services werden durch ein Netzwerk verfügbar und sichtbar gemacht und über ein definiertes Interfaces angesteuert [4]. Somit werden mehrere Einzelservices zu einem Dienst zusammengefasst. “Bei SOA handelt es sich somit um eine Struktur, welche die Unternehmensanwendungsintegration ermöglicht, indem die Komplexität der einzelnen Anwendungen („Applications“) hinter den standardisierten Schnittstellen verborgen wird“ [4]. Beispielsweise kann eine Bestellung ein Service sein, wobei die Anwendungen und Informationssysteme lediglich die Kenntnisse besitzen, dass der Dienst angeboten wird, welche Eingaben notwendig sind und wie die Rückgabe der Ergebnisse aussieht. Ein weiterer Vorteil von SOA ist die Wiederverwendbarkeit und die lose Kopplung der einzelnen Services untereinander.

Zu bemerken ist, dass es keine standardisierte Definition für SOA gibt, dennoch lassen sich idealtypische Eigenschaften von SOA auflisten. Diese Eigenschaften werden allerdings nicht von allen SOA vollständig eingehalten.

- Sichtbarkeit der Anforderungen und der Möglichkeiten eines Services.
- Ein Mittel zur Interaktion der Benutzer bieten.
- Lösung eines Problems der Benutzer bieten.
- Dienste sind abgeschlossen und können unabhängig voneinander eingesetzt werden.

- Dienste sind über ein Netzwerk verfügbar sein.

Grundsätzlich sollte ein Benutzer die Möglichkeit haben die Interaktionsmöglichkeiten mit dem System durch eine Dokumentation einzusehen. Darin sollten folgende Informationen bereit gestellt werden:

- Beschreibung aller verfügbaren Services. Meist durch ein Service Register zusammengefasst
- Die Funktionalität die ausgeführt wird
- Einschränkungen bei der Verwendung
- Eingabe und Ausgabe Formate. Darunter die verwendete Syntax und Semantik.
- Das verwendete Protokoll und den zu verwendenden Zugang (Internet oder privates Netzwerk)

SOA sind keine Webservices, da sie ein Architekturparadigma beschreiben, welches losgelöst von konkreten Implementierungstechniken und -Methoden ist [4].

2.3 Webservices

Durch ein Webservice können Anwendungen und Services über das Internet angesteuert werden und somit Funktionen ausgeführt und Informationen gelesen werden. Viele Webservices werden über das HTTP-Protokoll angesprochen. HTTP ist ein generisches, zustandsloses Protokoll zum Datentransfer über das Netzwerk. Durch dieses Protokoll werden Ressourcen ausgetauscht, die durch Uniform Resource Identifiers (URIs) identifiziert werden. Die ausgetauschten Ressourcen können sowohl statisch als auch dynamisch generiert werden. Zusätzlich besitzt jede zugängliche Ressource im Internet (bspw. ein Bild oder ein Video) eine Uniform Resource Locator (URL), welches diese Resource eindeutig referenziert. Zusätzlich liefern URLs den Namen des Protokolls, die Adresse (Domainname oder IP-Adresse) und den hierarchischen Pfad, um die bestimmte Ressource zu beziehen. Zudem werden in HTTP die Methoden OPTIONS, GET, POST, PUT und DELETE unterstützt [5].

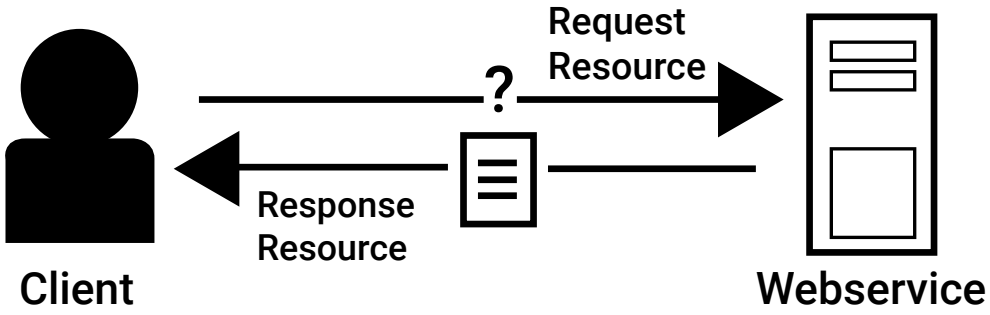

Abbildung 2.9: Schematische Darstellung einer Webservice-Anfrage

Ein Webservice ist eine Schnittstelle, die für andere Anwendungen in einem Netzwerk zugänglich ist. Dabei kann dieses Netzwerk sowohl das Internet als auch ein privates Netzwerk (Bsp. Firmennetzwerk) sein. Zusätzlich können zur Interaktion mit der Webservice Schnittstelle standardisierte Datenformate verwendet. Diese Interaktion wird durch ein internetbasiertes Protokoll unterstützt (siehe Abb.2.9) [5]. In der Abbildung 2.9 wird eine Anfrage an ein Webservice durch den Client über das Internet angestoßen, anschließend Antwort dieser Webservice dem Client.

2.3.1 Remote Procedure Calls (RPC)

RPC ist eine Technik mit dem Konzept, dass ein Client eine bestimmte entfernte Prozedur aufrufen kann und der Server das Programm, welches die Prozedur implementiert, aufruft. Die stärke von RPC war die saubere Art mit verteilten Anwendungen zu arbeiten. Durch RPC war es möglich verteilte Anwendungen zu starten ohne die Programmiersprache und das Programmiermuster zu ändern. Ursprünglich wurde RPC als eine Sammlung von Bibliotheken implementiert und später wurde es als eigene Plattform entwickelt. Heute ist RPC ein wichtiger Bestandteil vieler verteilter Informationssysteme [5].

2.3.2 Simple Object Access Protocol (SOAP)

SOAP ist ein Netzwerk- und Kommunikationsprotokoll, welches komplett auf HTTP basiert und Zustandslos ist. SOAP definiert wie Informationen in einer XML angeordnet und strukturiert sein müssen, damit es zwischen Anwendungen und Clients ausgetauscht werden kann [5]. Folgendes wird in SOAP spezifiziert:

- Ein Nachrichtenformat für die Ein-Weg-Kommunikation, indem beschrieben wird wie Informationen in einem XML Dokument verpackt werden sollen.
- Eine Sammlung an Konventionen für die Verwendung von SOAP-Nachrichten, die den RPC Interaktionspattern implementieren. Zusätzlich wird darin beschrieben wie Clients eine entfernte Prozedur aufrufen können indem eine SOAP-Nachricht gesendet wird und wie ein Service mit dem Senden einer SOAP-Nachricht antworten kann.
- Eine Sammlung an Regeln die jede Entität die eine SOAP-Nachricht verwendet befolgen muss. Genau wird definiert welche XML-Elemente eine Entität lesen und verstehen muss als auch die Aktionen die eine Entität unternehmen muss, falls der Inhalt nicht verstanden wird.
- Eine Beschreibung wie SOAP-Nachrichten über HTTP und SMTP transportiert werden sollen.

2.3.3 Representational State Transfer (REST)

REST ist die Abkürzung von Representational State Transfer, welches ein Programmierparadigma bezeichnet. REST wird in vielen Webservices wegen der Flexibilität, Skalierbarkeit und Performanz verwendet. REST wird auf Basis des HTTP-Protokolls verwendet und wird durch die Verwendung von Identifiers, Resourcen und Anwendungen sowie der Darstellung und den Datenformaten als auch den verwendeten Methoden beschrieben (siehe Abbildung 2.10) [6].

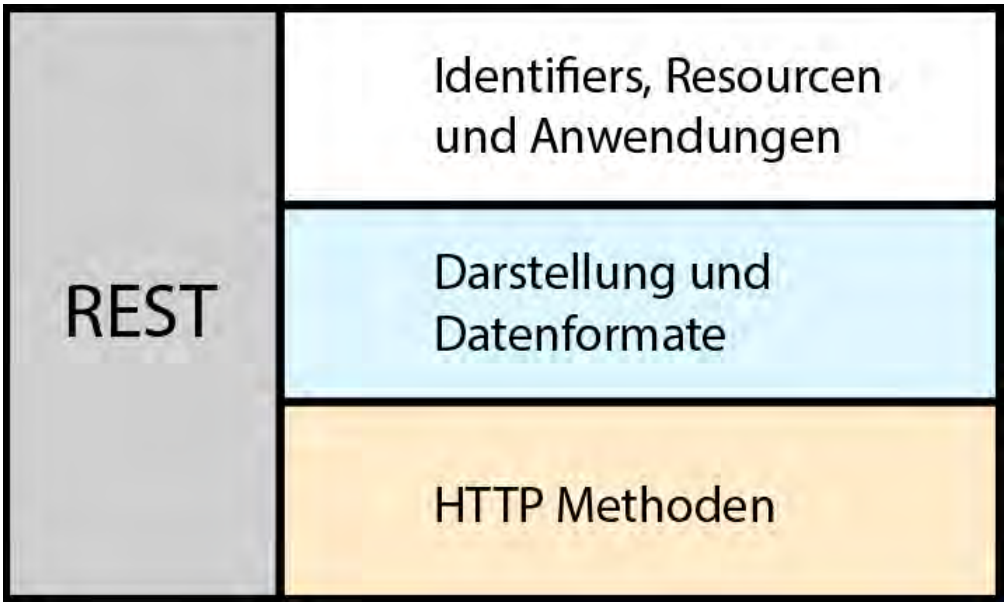

Abbildung 2.10: Schematische Darstellung von REST

Bei der Erstellung von REST-Services muss auf die Festlegung von Resourcenamen geachtet werden. Dabei spiegelt jede Resource eine Entität wieder, die durch den Namen identifiziert werden kann. Resourcen sind durch URIs definiert. Bei einer Anfrage entscheiden dann der entgegennehmende Server welche Entität die URI abbildet, damit die bestimmte Resource zurückgegeben werden kann. Eines der wichtigsten Bestandteile von REST ist die 'hypermedia as the engine of application state (HATEOAS)'. Hierbei werden Verlinkungen zwischen verschiedenen Resourcen generiert. Dadurch kann eine Anwendung oder ein Benutzer verfügbare Services erkennen und verwenden. Beispielsweise gibt es für einen REST Webservice nur einen Einstiegspunkt, nun muss die Anwendung anhand dieses Einstiegspunktes erkennen welche weiteren Services verfügbar sind [6].
Für die Datenübertragung müssen Client und Server ein bestimmtes Format und eine bestimmte Darstellung festlegen. Dies ist wichtig, damit die Kommunikation zwischen den Teilnehmern funktionieren kann. Hierbei ist es wichtig, dass die Wahl des Datenformats global und allgemein ist (Beispiel XML oder JSON) [6].
HTTP stellt die Basismethoden GET, PUT, POST und DELETE bereit. Dadurch können

REST-Services verschiedene Arten von Interaktionen durchführen. Somit kann GET und POST verwendet werden um Resourcen zu laden. Mit der PUT-Methode können Anwendungen und Clients Resourcen ersetzen oder neue Resourcen erstellen und mit der DELETE-Methode können Resourcen gelöscht werden [6].

2.4 Datenformate

Datenformate werden verwendet, damit bei der Kommunikation zwischen zwei Teilnehmern (Client, Server, Anwendungen) ein einheitliches Format verwendet wird. Im folgenden werden die Datenformate XML und JSON beschrieben.

2.4.1 Extensible Markup Language (XML)

XML ist eine Auszeichnungssprache. Dadurch können hierarchisch strukturierte Dateien erstellt werden. Im folgenden ist ein Beispiel für eine XML-Datei dargestellt.

```
1  <?xml version="1.0"?>
2  <books>
3          <book>
4                  <author>Max Mustermann</author>
5                  <ISBN>123-4-12332-102-1</ISBN>
6                  <publisher>Der-Verlag</publisher>
7          </book>
8          <book>
9                  <author>Alfred Mustermann</author>
10                 <ISBN>333-1-22222-101-1</ISBN>
11                 <publisher>Der-Verlag</publisher>
12         </book>
13 </books>
```

In der ersten Zeile wird ein XML-Dokument eingeleitet. Jedes XML-Dokument beginnt mit einem Root-Element, in diesem Beispiel '<books>'. Innerhalb dieses Elements können beliebig viele weitere Elemente existieren und hierarchisch geordnet werden.

2.4.2 JavaScript Object Notation (JSON)

Die JavaScript Object Notation (JSON) ist ein kompaktes Datenformat, das aufgrund seines geringen Container-Overheads oftmals für die Kommunikation zwischen Browsern und Webservices verwendet wird. Folgendes Beispiel stellt ein JSON-Objekt dar:

```
1 {
2         "type" : "book",
3         "author" : "Max Mustermann"
4 }
```

Ein JSON-Objekt wird durch Klammerung ('{ }') umschlossen. Die Attribute eines JSON-Objekts werden als Key-Value Paare beschrieben, dabei befindet sich der Key (Schlüssel) vor dem ':' und der Value (Wert) danach. Ein JSON-Objekt kann auch eine Liste an Objekten besitzen, wie im folgenden Beispiel dargestellt wird. Eine Liste wird durch eckige Klammern ('[ ]') definiert. Eine Liste kann beliebig viele Objekte oder weitere Listen beinhalten.

```
1 {
2         "type" : "book",
3         "author" : ["Max Mustermann", "Alfred Mustermann"]
4 }
```

2.5 Transaktionsmanagement

Da die zu entwickelnde Webservice Schnittstelle durch mehrere Benutzer und Anwendungen verwendet wird und die Prozessmodelle über die Schnittstelle bearbeitet werden

kann müssen Vorkehrungen zum Mehrfachbetrieb vorgenommen werden. Dabei können zwei bekannte Probleme auftreten:

Lost Update: Falls zwei Anwendungen ein bestimmtes Prozessmodell lesen. Die Anwendungen werden hierbei AW1 und AW2 genannt. Nachdem das lesen beendet wurde besitzen beide das selbe Prozessmodell. Nun bearbeiten sowohl AW1 als auch AW2 das Prozessmodell (AW1 zuerst), dabei überschreibt die Änderung des AW2 die Änderung des AW1. Dadurch ist die Änderung des AW1 verloren gegangen.
Dirty Read: Hierbei wird erneut angenommen, dass 2 Anwendungen (AW1 und AW2) auf ein Prozessmodell zugreifen wollen. Dabei liest AW1 zuerst das Prozessmodell und ändert Ihn ab. Nun ließt AW2 das geänderte Prozessmodell und arbeitet mit diesen. Allerdings wird die Änderungen die AW1 durchgeführt hatte rückgängig gemacht. Nun arbeitet allerdings AW2 mit falschen Daten.

Zusätzlich zu der Thematik greift auch ein weiterer Aspekt an, das CAP-Theorem (siehe [7]). Das CAP-Theorem besagt, dass ein Netzwerksystem nur 2 der 3 Eigenschaften zufriedenstellend implementieren kann. Die 3 Eigenschaften sind:

- Konsistenz (C): Nach jeder Transaktion wird sichergestellt, dass die Daten in einem Konsistenten Zustand übernommen werden.
- Verfügbarkeit (A): Das System kann die Anfragen entgegennehmen und beantworten.
- Partitionstoleranz (P): Die Daten können partitioniert an verschiedenen Orten gespeichert werden.

Die Problematik hierbei ist, je konsistenter das System ist, desto weniger verfügbar ist es, da bei jeder Transaktion sichergestellt werden muss, dass das System in einem konsistenten Zustand überführt wird und in der Zeit eine erhöhte Antwortdauer entsteht.

3

Entwurf einer Webservice Schnittstelle für Prozessmanagementsysteme

3.1 Einleitung

Um die Effizienz und Effektivität in einem Unternehmen zu steigern werden Geschäftsprozesse mithilfe von Prozessmanagement erhoben, analysiert, optimiert und überwacht. Durch die Verwendung von Prozessmanagementsystemen wird die systemgestützte Ausführung von Prozessmodellen unterstützt und somit automatisierte Aktivitäten ermöglicht. Um die Effizienz und die Bedienbarkeit zu erhöhen werden angepasste Oberflächen und Applikationen für die Ausführung von Aktivitäten verwendet. Dadurch müssen sowohl Informationssysteme als auch Applikationen mit einem Prozessmanagementsystem kommunizieren können. Um diese unterschiedlichen Informationssysteme anzubinden, bietet sich die Entwicklung einer generischen Webservice Schnittstelle an. Dadurch wird die Kommunikation über eine standardisierte Webservice Schnittstelle mit dem Prozessmanagementsystem realisiert und ermöglicht somit eine einheitliche Einbindung der einzelnen Informationssysteme und Applikationen. Einige Prozessmanagementsysteme, die diesen Ansatz bereits verfolgen, werden im Kapitel 6 beschrieben und betrachten. Allerdings wird bei diesen Ansätzen nicht der gesamte Prozesslebenszyklus abgedeckt, sondern nur Teilaspekte, wie beispielsweise die Modellierung von Prozessmodellen oder die Ausführung einzelner Aktivitäten.

Da Unternehmen oftmals mit mehreren Prozessmanagementsystemen arbeiten und verschiedene Informationssysteme und Applikationen mit verschiedenen Prozessmanagementsystemen arbeiten müssen, wird eine einheitliche Webservice Schnittstelle mit

einer einheitlichen Darstellungen von Prozessdaten und Methoden zur Anpassung einelner Prozessaspekte, z.B. die Ausführung von Prozessinstanzen, benötigt. Zudem sollte solch eine Webservice Schnittstelle idealerweise den kompletten Prozesslebenszyklus abdecken können.

In diesem Kapitel wird eine Webservice Schnittstelle für Prozessmanagementsysteme entworfen, die alle relevanten Aktivitäten des Prozesslebenszyklus abdeckt und eine einheitliche Darstellung der Schnittstelle sowie der Prozessdaten darstellt. Zusätzlich werden Anforderungen an ein in diesem Zusammenhang benötigtes *Request-Response-Protokoll* definiert und die Architektur des Webservices näher betrachtet. Anschließend wird die Schnittstelle zum Prozessmanagementsystem genauer beschrieben.

3.2 Anwendungsfälle

In diesem Abschnitt werden drei verschiedene Anwendungsfälle vorgestellt, mit deren Hilfe die im nachfolgenden Abschnitt erstellten Anforderungen an eine Webservice Schnittstelle erarbeitet werden. Alle Anwendungsfälle sind mithilfe der BPMN 2.0 Notation dargestellt und beschreiben jeweils ein mögliches Szenario der Webservice Schnittstelle.

3.2.1 Patientenaufnahme in einem Krankenhaus

Das erste Prozessmodell aus Abbildung 3.1 beschreibt einen vereinfachten Geschäftsprozess zur Patientenaufnahme in einem Krankenhaus. Dabei wird in diesem Prozessmodell zunächst der Patient aufgenommen, dabei wird diese Tätigkeit durch eine Person an der Rezeption durchgeführt. Der behandelnde Arzt führt anschließend in einer Besprechung mit dem Patienten gleichzeitig eine Anamnese durch und legt die dadurch erforderlichen Untersuchungen fest. Anschließend werden die festgelegten Untersuchungen durchgeführt. Dabei können eine, mehrere oder keine Untersuchung durchgeführt werden, diese Entscheidung wird vom Arzt festgelegt. Anschließend wird eine Diagnose erstellt und die Therapie festgelegt. Die Auswahl keiner Untersuchung macht in dem Fall Sinn,

falls eine Diagnose ohne weitere Untersuchungen erstellt werden kann. Am Ende des Prozesses muss die Rezeption entweder eine Überweisung, ein Rezept oder Beides ausstellen, wobei auch in diesem Schritt keine Therapie auswählbar sein kann.

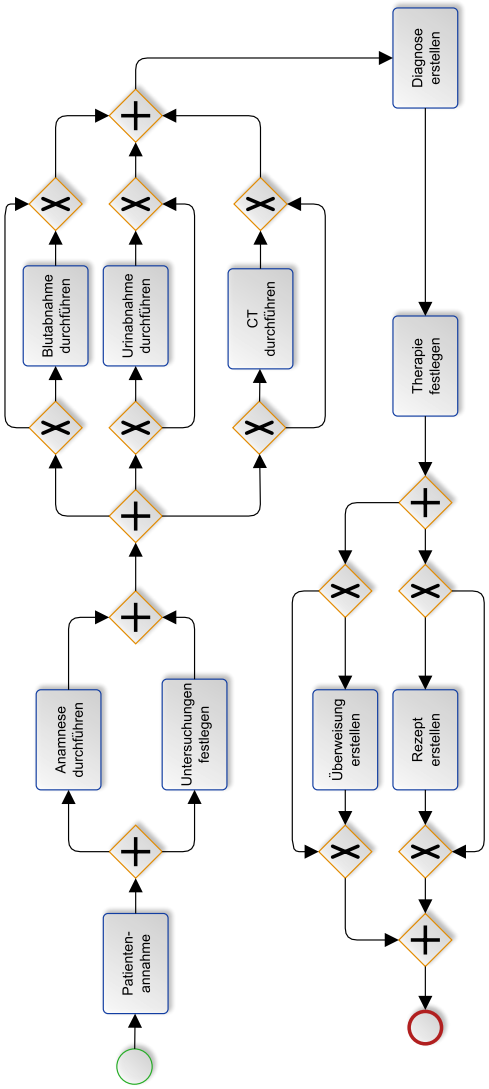

Abbildung 3.1: Vereinfachtes Prozessmodell einer Patienaufnahme

Analyse und Anforderungen: Dieses Prozessmodell spiegelt die Patientenaufnahme und der Ablauf der Diagnose im Normalfall wieder. Allerdings existieren in einem Krankenhaus bestimmte Fälle, bei denen eine spezifische Behandlung oder Untersuchung

anfällt, die nicht durch das Prozessmodell abgedeckt wird. Deshalb muss das Prozessmodell bzw. eine Instanz bei dessen Ausführung auch *bearbeitbar* und *anpassbar* sein. Zusätzlich muss der behandelnde Arzt einen bestimmten Patienten bzw. eine bestimmte Instanz *suchen* und *filtern* können. Da verschiedene Benutzer an verschiedenen Orten darauf zugreifen müssen, sollte ein implementiertes Informationssystem über ein Netzwerk verfügbar sein. Zieht man zusätzlich in Betracht, dass jeder Benutzer auf eine individuelle Benutzeroberfläche zugreift, so muss die Webservice Schnittstelle ein allgemeines, plattformunabhängiges Standardformat für den Datenaustausch bereitstellen. Ein weiterer wichtiger Aspekt in einem Krankenhaus ist, dass verschiedene Mitarbeiter mit verschiedenen Anwendungen auf ein oder mehrere Prozessmanagementsysteme zugreifen müssen. Daher sollte die Webservice Schnittstelle die Möglichkeit bieten, mehrere verschiedene Prozessmanagementsysteme und Informationssysteme über eine Webservice Schnittstelle kommunizieren zu lassen.

3.2.2 Prozess im maschinellen Umfeld

Dieses Prozessmodell stellt einen Teilprozess einer Verpackungsmaschine für Tabletten dar. Dieser Teil der Maschine ist für die Einsetzung der Tabletten und für die Versiegelung des Blisters zuständig. In Abbildung 3.2 ist dieses Prozessmodell vereinfacht dargestellt. Bei jedem Verpacken der Tabletten benötigt die Maschine Herstellungsparameter in Form von Daten der zu verpackenden Tabletten und deren Verpackung sowie Herstellungsparameter der Maschine (Temperaturen, Zyklenzeiten, Materialbeschaffenheiten). Anschließend heizt die Maschine die Walze für die Versiegelung vor, diese muss je nach gewähltem Verpackungsmaterial eine bestimmte Temperatur besitzen. Hierbei wird gleichzeitig das Förderband solange bewegt, bis ein Blister eine Lichtschranke erreicht. Sobald ein Blister an der Lichtschranke ankommt, wird das Förderband angehalten und der Blister mit Tabletten befüllt. Nachdem der Blister befüllt worden ist und die Walze vorgeheizt wurde, wird die Versiegelungsfolie aufgelegt und mit dem Blister versiegelt.

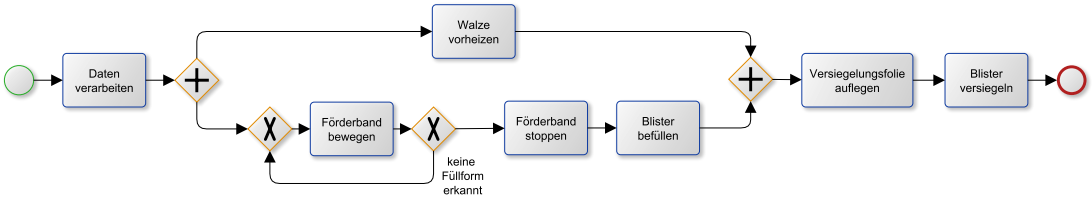

Abbildung 3.2: Prozessmodell einer Teilmaschine zur Verpackung von Tabletten - Einsetzen von Tabletten und verschließen

Analyse und Anforderungen: Hierbei handelt es sich um einen stark automatisierten Prozess, der durch eine Maschine ausgeführt wird. Dabei muss die Maschine eine Datenschnittstelle verwenden, um den Zustand der Ausführung einem anderen Informationssystem (beispielsweise SCADA- oder PPS-Systeme) mitteilen zu können. Die Daten der Ausführung und der Zustand jeder Prozessinstanz (in unserem Beispiel wird für jeden Blister eine eigene Prozessinstanz betrachtet) kann von einem Techniker überwacht werden. Dadurch kann beispielsweise festgestellt werden, ob ein Prozess zu lange läuft oder ob eine Maschine stehen geblieben ist. Da ein Techniker im Regelfall mehrere Geräte überwachen muss, sollten Schnittstellen für eine Art *Dashboard* bereitgestellt werden. Zusätzlich muss die Maschine automatisiert mit einer Datenschnittstelle arbeiten können. Aus diesem Grund sollten die Funktionen der Datenschnittstelle durch die Maschine erkennbar sein. Da in unserem Fall die Datenschnittstelle durch eine Webservice Schnittstelle implementiert werden soll, können diese Anforderungen analog betrachtet werden.

3.2.3 Bestellprozess eines Online-Versandhauses

In Abbildung 3.3 ist der Bestellprozess eines Online-Versandhauses abgebildet. In diesem Prozessmodell beginnt der Kunde damit Artikel in den Warenkorb zu legen; dabei kann dieser Kunde so viele Artikel wie benötigt einkaufen. Ist der Kunde fertig, bestätigt dieser seinen Einkauf (abgebildet durch die Aktivität 'Zur Kasse gehen'). Anschließend wird die Bestellung vom Versandhaus verarbeitet. Darauf folgend werden die Artikel reserviert sowie gleichzeitig eine Rechnung erstellt und an den Kunden gesendet. Der

Kunde muss dann die Rechnung bezahlen, bevor dessen Zahlung verbucht werden kann. Erst nachdem der Kunde gezahlt hat und die Artikel reserviert wurden kann mit der Lieferung begonnen werden. Sobald der Kunde das Paket entgegennimmt, ist der Prozess beendet.

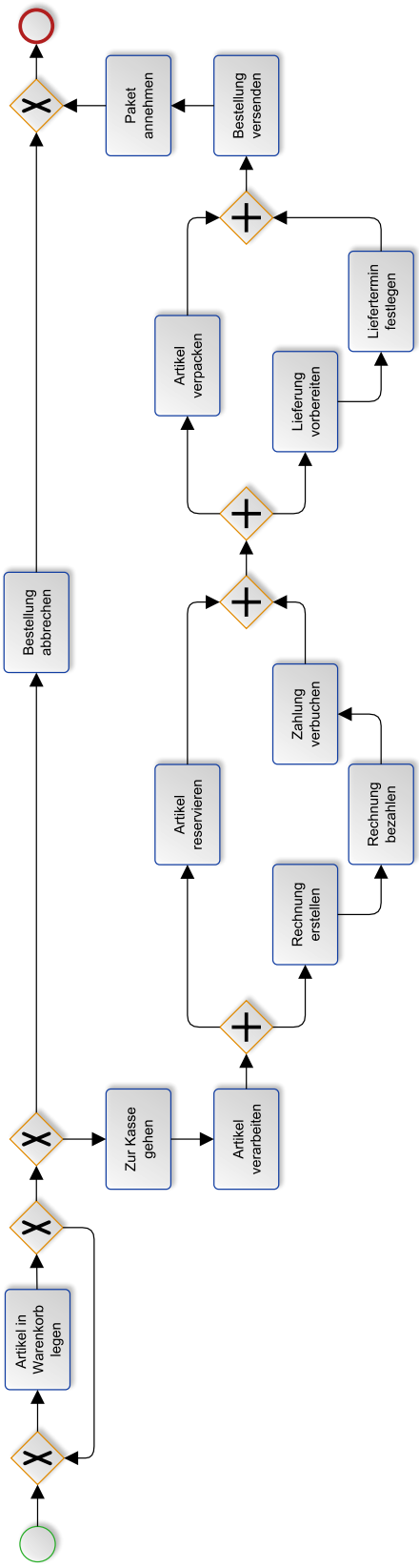

Abbildung 3.3: Bestellprozess auf einer Online-Plattform

Analyse und Anforderungen: Da der Kunde verschiedene Möglichkeiten zur Bestellung besitzt (Webshop, mobile Applikation oder mobile Webseite), muss eine Webservice Schnittstelle Anfragen plattformunabhängig verarbeiten können und mit einem geeigneten Datenformat arbeiten können. Da im Versandhaus verschiedene Mitarbeiter am Bestellprozess involviert sind, muss die Schnittstelle die Aufgaben auf bestimmte Mitarbeiter filtern können. Während einer Bestellung in einem Online Versandhaus kann es vorkommen, dass der Kunde eine Ware fälschlicherweise bestellt hat oder die Bestellung aus anderen Gründen stornieren möchte. Deswegen kann der Kunde zusätzlich, wie in Abbildung 3.4 dargestellt, eine Bestellung auch stornieren (falls der Bestellvorgang noch nicht zu weit fortgeschritten ist). Das bedeutet, dass eine Prozessinstanz einer Bestellung von einem Mitarbeiter des Online-Versandhauses abgebrochen werden kann, falls diese storniert wird. Deshalb muss die Schnittstelle Funktionen zur Erstellung, Manipulation und Beendigung von Prozessinstanzen bzw. deren Prozessmodelle bereitstellen.

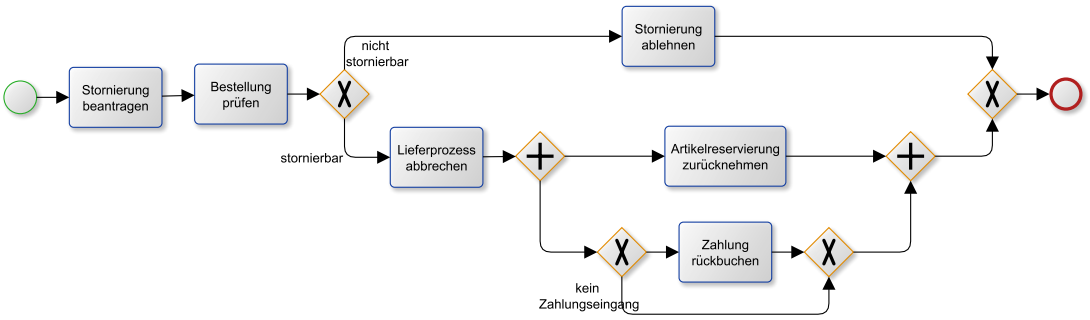

Abbildung 3.4: Stornierungsprozess einer Bestellung

3.3 Anforderungen

In diesem Abschnitt werden die Anforderungen der Webservice Schnittstelle sowohl technisch als auch funktional analysiert und definiert.

3.3.1 Funktionalen Anforderungen

Funktionale Anforderungen beschreiben Funktionen, welche die zu entwickelnde Webservice Schnittstelle zur Verfügung stellen sollte. Diese Funktionen bzw. Anwendungsfälle werden in Abbildung 3.5 dargestellt. Zusätzlich zu den Anforderungen aus dem Anwendungsfalldiagramm sollte die Webservice Schnittstelle einen Login bereitstellen oder eine Möglichkeit zur Authentifizierung von Benutzern, Sicherheit bei der Ausführung der einzelnen Funktionen und der Rechteverwaltung bieten. Außerdem sollte die Webservice Schnittstelle zu bestehenden Systemen, beispielsweise der Clavii BPM Platform (siehe Kapitel 4), integrierbar sein. Dies bedeutet somit auch, dass die Schnittstelle auch in bestehende Systeme integriert werden kann. Ein weiterer Punkt der funktionalen Anforderungen ist, dass die Webservice Schnittstelle durch verschiedene Anwendungen verwendet werden und mit verschiedenen Prozessmanagementsystemen kommunizieren können sollte.

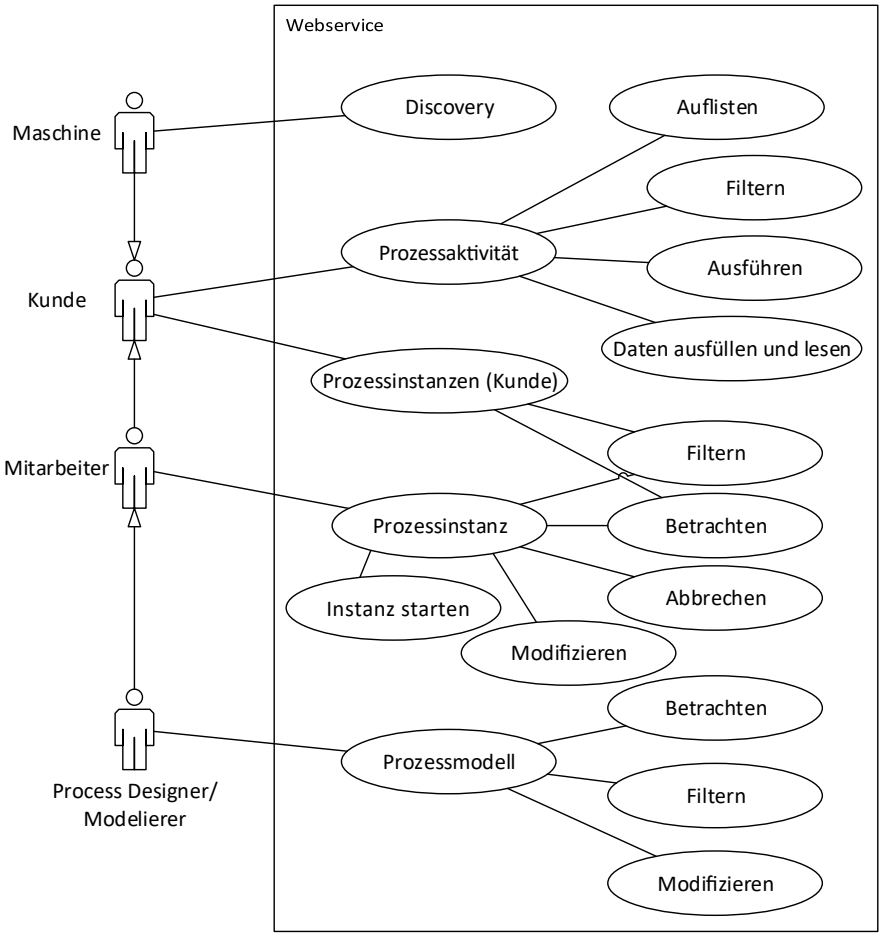

Abbildung 3.5: Anwendungsfalldiagramm zur Funktionalität des Webservice

Im Anwendungsfalldiagramm aus Abbildung 3.5 werden die verschiedenen Akteure ’Maschine’, ’Kunde’, ’Mitarbeiter’ und ’Process Designer’ dargestellt (siehe Abbildung 3.5 links). Diese Akteure spiegeln die in den Szenarien beschriebenen Akteure wieder. Somit beinhaltet der ’Kunde’ den ’Kunden’ selbst und den ’Process Participant (Prozessbeteiligter)’. Der ’Mitarbeiter’ hingegen beinhaltet den ’Process Participant’, ’Business Engineer’ und den ’Knowledge Worker’. Der Akteur ’Process Designer’ beinhaltet den Stakeholder ’Process Designer’, ’Chief Process Officer’, ’Process Responsible’ und den ’System Architekt’. Der Akteur ’Maschine’ umfasst alle ’automatisierten Maschinen und Geräte’, die mit einem Prozessmanagementsystem interagieren sollen. Zusätzlich sollte die funktionale Hierarchie der einzelnen Akteure betrachtet werden; dabei besitzt der Process Designer/Modelierer alle möglichen Funktionalitäten der Schnittstelle, wohingegen der Kunde nur beschränkte Funktionalität besitzt, beispielsweise zur Modifikation seiner eigenen Prozessinstanzen. Tabelle 3.1 führt alle funktionalen Anforderungen auf.

Tabelle 3.1: Funktionale Anforderungen - Teil 1

Bezeichner	Anforderung	Funktion	Beschreibung
F1	Discovery		Dieser Anwendungsfall ist für alle Maschinen und Geräte notwendig, um mit der Webservice Schnittstelle zu kommunizieren. Hierbei ist es wichtig, dass die Schnittstelle Methoden zur Verfügung stellt, mit denen die Funktionalität des Webservices automatisch erkannt und verwendet werden kann.
F2	Prozessausführung	Prozessinstanzebene und Aufgabenlistenebene	Funktionen zur Ausführung von Prozessinstanzen oder Aktivitäten
F2.1	Worklistebene	Auflisten, Filtern, Ausführen, Daten auslesen und ausführen	Die Aktivitäten sind weiter in den Anwendungsfällen 'Aktivitäten auflisten', 'Aktivitäten filtern', 'Aktivitäten ausführen' und 'Daten der Aktivitäten auslesen' sowie '-ausfüllen' aufgeteilt.
F2.2	Prozessinstanzebene	Filtern, Betrachten, Abbrechen, Modifizieren, Instanz starten	Diese Funktionalität beschreibt die Möglichkeit mit Prozessinstanzen zu arbeiten. Dabei kann ein Prozessbeteiligter gefiltert Prozessinstanzen lesen und auch starten können, wobei ein anderer Prozessbeteiligter zusätzlich eine Prozessinstanz abbrechen und modifizieren kann.

Tabelle 3.2: Funktionale Anforderungen - Teil 2

Bezeichner	Anforderung	Funktion	Beschreibung
F3	Prozessmodellierung	Betrachten, Filtern, Modifizieren, Instanz starten	Die Anwendungsfälle wurden in die Funktionen 'Prozessmodelle betrachten', 'Prozessmodelle filtern' und 'Prozessmodelle modifizieren' aufgeteilt. Diese Interaktion mit dem System ist wichtig um Prozessmodelle zu verbessern oder zu erweitern
F4	Authentifizierung	Login, Logout, Authentifizierung, Rollenmanagement	Ein Benutzer sollte sich gegenüber einem System authentifizieren. Zusätzlich sollten Benutzer einer bestimmten Rolle zugeordnet werden können.
F5	Prozessüberwachung	Log-Daten auslesen, Metriken ausgeben	Diese Funktionen werden beispielsweise für das Monitoring laufender Prozessinstanzen benötigt. Hier können Informationen, wie beispielsweise die durchschnittliche Ausführungszeit einer Prozessinstanz, abgerufen werden.

3.3.2 Nicht-funktionale Anforderungen

In diesem Abschnitt werden nicht-funktionale Anforderungen aufgelistet, welche die Webservice Schnittstelle unterstützen sollte. Die Anforderungen definieren und grenzen die für eine prototypische Implementierung zur Verfügung stehenden Protokolle, Programmiersprachen und Programmierparadigmen ein. Tabelle 3.3 listet die technischen Anforderungen der Webservice Schnittstelle bzw. dessen Implementierung auf.

Tabelle 3.3: Nicht-funktionale Anforderungen

Bezeichner	Anforderung	Beschreibung
T1	Verfügbarkeit	Die Schnittstelle sollte durch ein Netzwerk global verfügbar sein und über verschiedene Anwendungen angesteuert werden können.
T2	Standardisierung	Da die Schnittstelle durch verschiedene Geräte ansteuerbar sein sollte, sollte das zu verwendende Protokoll standardisiert sein. Zusätzlich sollte eine einheitliche Protokolldarstellung gewährleistet werden.
T3	Verständlichkeit	Eine Programmierung gegen die Webservice Schnittstelle sollte einfach und leicht verständlich sein.
T4	Sicherheit	Die Sicherheit einer Datenübertragung zwischen Informationssystem und Webservice Schnittstelle sollte gewährleistet werden.
T5	Zuverlässigkeit und Fehlertoleranz	Die Übertragung der Daten und Ausführung sollte zuverlässig stattfinden. Zusätzlich sollten Fehler bei der Übertragung erkannt und behoben werden können.

3.4 Lösungskonzept

In diesem Abschnitt wird anhand der oben aufgeführten Anforderungen ein Lösungskonzept für eine prozessorientierte Webservice Schnittstelle vorgestellt. Dabei werden die drei Aspekte *Protokoll*, *Webservice Schnittstelle* und *Transformation* betrachtet. Der Aspekt *Protokoll* (siehe Abbildung 3.6a) beschreibt das Protokoll der Schnittstelle, die durch Anwendungen und Informationssystemen verwendet werden und in Kapitel 3.4.1 genauer untersucht wird, der Aspekt *Webservice Schnittstelle* (siehe Abbildung 3.6b) beschreibt die Architektur der Webservice Schnittstelle und wird in Kapitel 3.4.2 definiert. Der Aspekt *Transformation* (siehe Abbildung 3.6c) befasst sich mit der Schnittstelle zu einem Prozessmanagementsystem und wird in Kapitel 3.4.3 näher beschrieben.

Abbildung 3.6: Komponenten des Lösungskonzepts

3.4.1 Request-Response-Protokoll

In diesem Abschnitt wird das Standardprotokoll, welches zur Datenübertragung zwischen Informationssystem (Anwendung) und der Webservice Schnittstelle verwendet wird, genauer beschrieben. Das Protokoll sollte aufgrund der technischen Anforderungen *T1* und *T2* (siehe Tabelle 3.3) ein standardisiertes Netzwerkprotokoll sein, damit dieses von verschiedenen Anwendungen verwendet werden kann. Zusätzlich sollte das Protokoll aufgrund der technischen Anforderung *T4* und der funktionalen Anforderung *F4* (siehe Tabelle 3.2) Sicherheit bei der Datenübertragung und eine Möglichkeit zur Authentifizierung bieten. Dabei können Authentifizierungsmerkmale über das Protokoll mitgesendet werden. Ein weiteres wichtiges Merkmal des Protokolls ist die Anforderung *T5*: das Protokoll sollte eine zuverlässige Übertragung der Anfragen gewährleisten und gleichzeitig bei Verlust einer Anfrage oder einer Antwort die Anfrage erneut ver-

senden können. Aufgrund der Anforderungen *T2*, *T3* und *F2-F6* (siehe Tabellen 3.1 und 3.2) sollten angeforderte Daten, wie z.B. Prozessmodelle, Prozessinstanzen und Aktivitäten, eine einheitlichen Darstellung zugrunde liegen und über ein standardisiertes Datenformat übertragen werden. Dadurch können Anwendungen mit verschiedenen Prozessmanagementsystemen kommunizieren, ohne technisch spezifische Darstellungen von Prozessmodellen oder Prozessinstanzen betrachten zu müssen (siehe hierzu das Beispiel des Routing Point Providers aus Abbildung 3.4.2).

3.4.2 Architekturschema

Die vorgestellten Funktionen sowie die zugrundeliegende Softwarearchitektur der Webservice Schnittstelle lassen sich zu folgenden Schema gruppieren (siehe Abbildung 3.7): die Hauptpakete der Architektur sind der *DiscoveryController*, *IdentityController*, *TaskController*, *ModelController* und *InstanceController*.

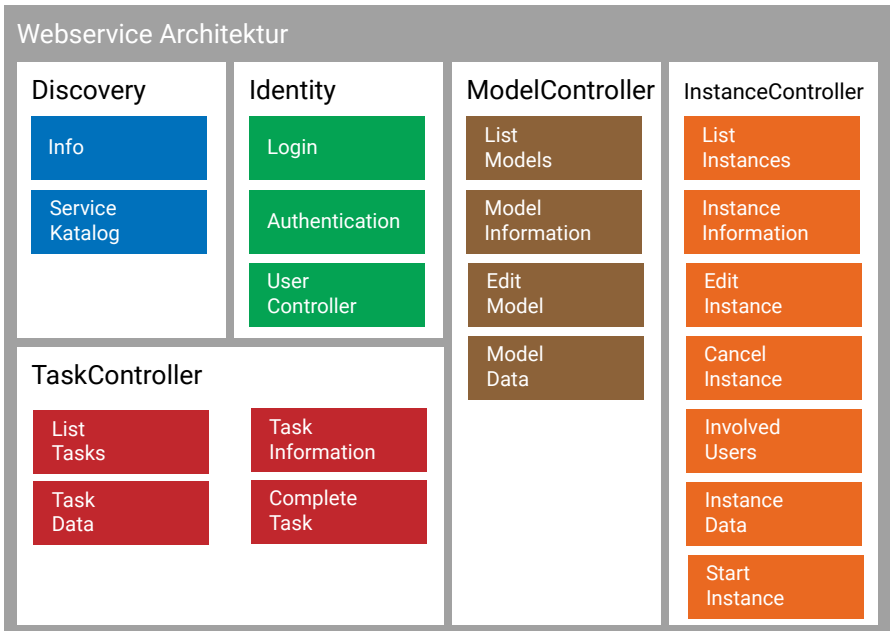

Abbildung 3.7: Architektur der Webservice Schnittstelle

Das Paket *DiscoveryController* (siehe Abbildung 3.7) umfasst die Services *Info* und *ServiceKatalog*, und deckt die Anforderung *F1* ab (siehe Tabelle 3.1). Der Service *Info*

gibt Informationen zur Webservice Schnittstelle zurück, dadurch wird die Anwendung beispielsweise über Name und Version der Schnittstelle benachrichtigt. Der *Service Katalog* ist für die Funktionsbeschreibung der Schnittstelle zuständig. Dabei werden die verschiedenen verfügbaren Services mit Beschreibung, URL und Parameter zurückgegeben.

Die Services *Login*, *Authentication* und *UserController* werden durch den *IdentityController* gruppiert und decken Anforderung *F4* ab (siehe Tabelle 3.2). Der Service *Login* stellt eine Möglichkeit für die Authentifikation eines Benutzers bereit. Dabei wird ein Benutzer anhand seiner Benutzerdaten eingeloggt und ein spezieller Authentifizierungskey (englisch: *API key*) generiert. Die Authentifizierung wird durch den *Authentication* Service behandelt. Der letzte Service im Paket ist der *UserController*. Dieser besteht aus einer Sammlung von Funktionen, um Benutzer zu suchen, Benutzer anzusehen und Rollen der Benutzer zu betrachten bzw. zu filtern. Zusätzlich stehen Funktionen bereit, um bestimmte Benutzer einer Rolle abzurufen. Das Paket TaskController beinhaltet die Services *ListTasks*, *TaskInformationen*, *TaskData* und *CompleteTask*. Durch den *ListTasks* Service werden die eigenen Prozessaktivitäten oder die Prozessaktivitäten eines bestimmten Benutzers als Liste zurückgegeben. Dabei lässt sich die Rückgabe an bestimmten Parametern filtern. Der *TaskInformation* Service bietet Möglichkeiten um die Informationen einer spezifischen Prozessaktivität zu erhalten. Damit Prozessaktivitäten auch beendet und ausgeführt werden können, ist der Service *CompleteTask* zuständig. Da jede Prozessaktivität auch Daten lesen oder schreiben kann wird ein *TaskData* Service bereitgestellt, dieser besitzt Funktionen zum Lesen oder Schreiben der Daten. Dadurch deckt das Paket TaskController die Anforderung F2.1 abgedeckt. Die Services *ListModels*, *ModelInformation*, *EditModel* und *ModelData* werden im Paket ModelController zusammengefasst und decken die Anforderung F3 ab. Durch den *ListModels* Service lassen sich Prozessmodelle in einem Prozessmanagementsystem filtern und auflisten. Hierbei ist die Prozessmodellbezeichnung ein möglicher Filter, da die Suche anhand der Bezeichnung für Prozessbeteiligte einfacher ist. Der *ModelInformation* Service bietet zudem Funktionen zur Verfügung um die Informationen eines Prozessmodels und deren Aktivitäten einzusehen. Alle Prozessmodelle lassen sich durch den *EditModel* Service bearbeiten. Dies bedeutet, dass es möglich ist neue Akti-

vitäten einzufügen oder Aktivitäten zu entfernen. Die verschiedenen Daten, die in einem Prozessmodell gespeichert werden, können anschließend mit dem *ModelData* Service abgerufen werden.

Das letzte Paket in der Abbildung 3.7 (rechts) beinhaltet die Services *ListInstances*, *InstanceInformation*, *EditInstance*, *StartInstance*, *CancelInstance*, *InvolvedUsers* und *InstanceData*. Durch die Services *ListInstances* und *InstanceInformation* lassen sich die Prozessinstanzen filtern und als Liste ausgeben sowie die Informationen zu diesen Prozessinstanzen anzeigen. Beispiele möglicher Filterattribute sind: Prozessmodell, Startzeit, Endzeit, Prozessbeteiligter. Durch den *EditInstance* Service lassen sich Prozessinstanzen bearbeiten und durch *CancelInstance* abbrechen. Zusätzlich besteht die Möglichkeit, mithilfe des *InvolvedUsers* Service alle Prozessbeteiligten abzurufen, die an dieser Prozessinstanz involviert sind. Datenelemente, die während der Ausführung der Prozessinstanz geschrieben oder gelesen werden sollen, können mithilfe des *InstanceData* Service zurückgegeben werden. Durch den *StartInstance* Service lässt sich eine Instanz eines Prozessmodells starten. Dieses Paket deckt dabei Anforderung F2.2 ab.

Nach Betrachtung der Webservice Schnittstellenarchitektur wird im folgenden Kapitel ein Schnittstellenmodell erstellt, mit dessen Hilfe der Aufbau der Webservice Schnittstelle genauer beschrieben werden kann.

Schnittstellenmodell

Nachdem in den vorherigen Kapitel die Funktionen und Services beschrieben worden sind, wird das Schnittstellenmodell der zu entwickelnden Webservice Schnittstelle näher betrachtet. Abbildung 3.8 zeigt eine vereinfachte Darstellung der Webservice Schnittstelle.

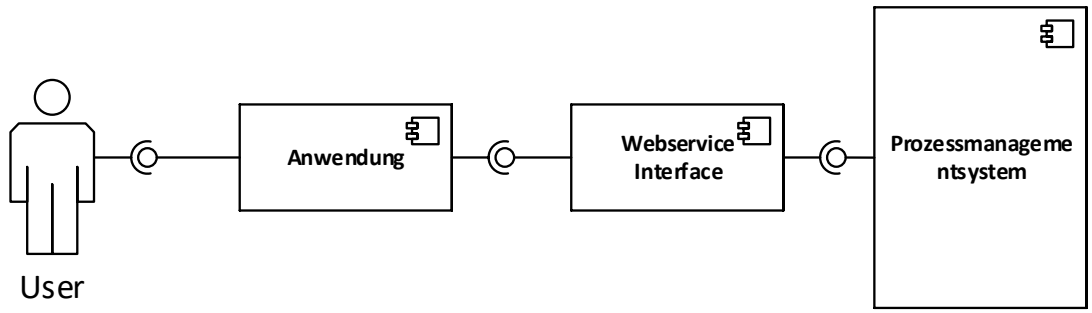

Abbildung 3.8: Komponentenmodell einer Webservice Schnittstelle

Der Schnittstellenbenutzer, in Abbildung 3.8 links als 'User' gekennzeichnet, verwendet typischerweise eine Anwendung, die mit einem Prozessmanagementsystem kommunizieren möchte. Damit die Anwendung auf die Daten des Prozessmanagementsystems zugreifen kann, wird die vorgestellte Webservice Schnittstelle angesteuert. Daten der Schnittstelle werden durch ein standardisiertes Datenformat, wie beispielsweise JSON oder XML, versendet und empfangen. Um alle Anforderungen wie in Kapitel 3.3 beschrieben abzudecken, wurde die Webservice Schnittstelle in den Komponenten *Basic Service Provider*, *Service Discovery Provider* und *Routing Point Provider* aufgeteilt.

Basic Service Provider

Der Basic Service Provider (BSP) ist die Grundstruktur der Webservice Schnittstelle und definiert den Ablauf der Kommunikation mit einem bestimmten Service der Schnittstelle. Abbildung 3.9 zeigt die Komponenten des Basic Service Providers.

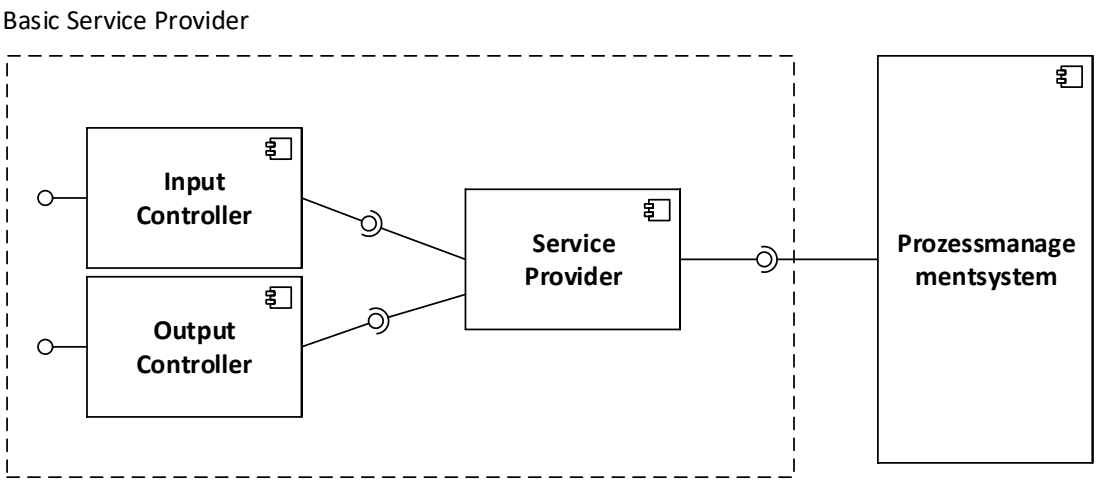

Abbildung 3.9: Komponentenmodell des Basic Service Providers

Diese bestehen aus dem *InputController*, *OutputController* und dem *ServiceProvider*. Dabei verarbeitet der *InputController* alle eingehenden Anfragen, verarbeitet die empfangenen Daten und entscheidet, welcher ServiceProvider hierzu verwendet werden muss. Der *ServiceProvider* stellt einen Service dar, welcher in Kapitel 3.4.2 definiert wurde. Nachdem eine Anfrage bearbeitet worden ist werden die angeforderten Daten mit Hilfe des *OutputController* zurückgesendet. In Abbildung 3.10 ist ein Beispiel zur Abfrage von Prozessmodellen zu sehen, die einem Benutzer gehören oder auf die er Zugriff hat.

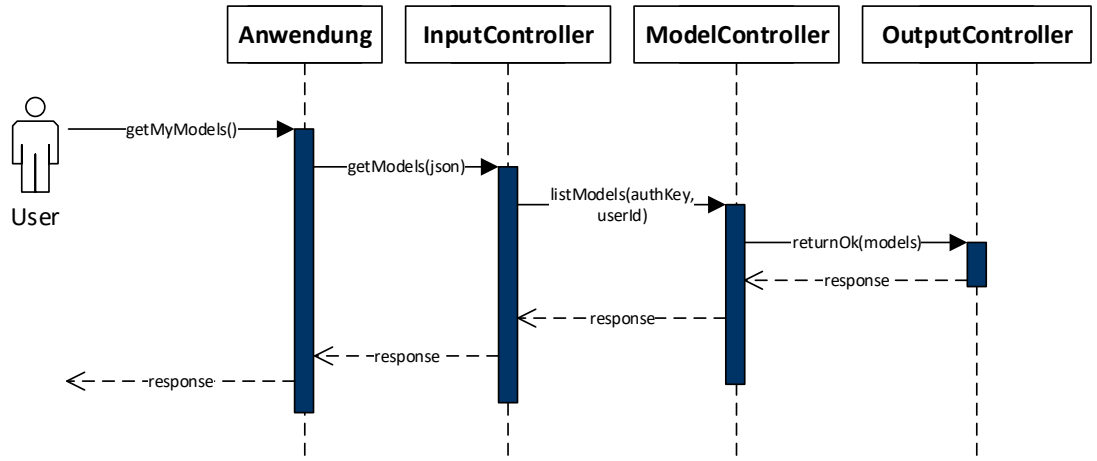

Abbildung 3.10: Sequenzdiagramm des Basic Service Providers

Der Schnittstellenbenutzer (siehe Abbildung 3.10) oder ein Informationssystem tätigt die Anfrage bestimmte Prozessmodelle zu betrachten. Die Anwendung fordert die nötigen Daten durch den Webservice an und übergibt die geforderten Daten in einem standardisierten Datenformat (in unserem Beispiel JSON) zurück. Die eingehende Anfrage wird vom InputController überprüft und an den ModelController weitergeleitet. Nachdem die nötigen Daten durch den Service gesammelt wurden, werden diese mit Hilfe des OutputController an die Anwendung zurück gesendet.

Service Discovery Provider

Der Service Discovery Provider (SDP) deckt die funktionale Anforderung *F1* (siehe Tabelle 3.1) ab. Mithilfe des Service Discovery Provider lässt sich der Funktionsumfang des zugrundeliegenden Prozessmanagementsystems feststellen und über diesen Provider abfragen. Dadurch können Anwendungen oder Client-APIs die zur Verfügung stehenden Services abfragen und ansteuern. Zusätzlich werden Informationen über notwendige Parameter und Beschreibung der angefragten Services bzw. Funktionen dargelegt. Abbildung 3.11 zeigt den Komponenten-Aufbau des Service Discovery Providers.

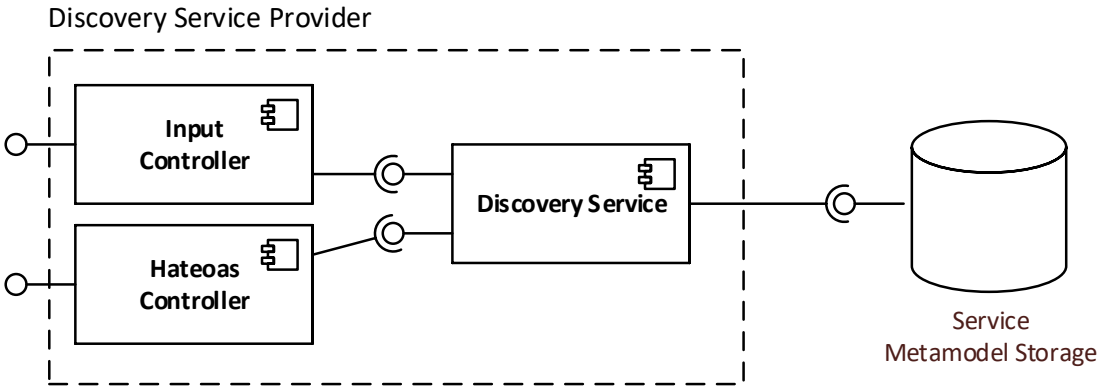

Abbildung 3.11: Komponentenmodell des Service Discovery Providers

Dieser verwendet zur Verarbeitung der Anfrage ebenfalls die InputController Komponente. Die Service Discovery Komponente verarbeitet die Anfrage und sendet eine Menge oder eine einzelne Information zum Service zurück. Die Informationen oder Daten der Services kann in einer Datenbank oder in einer anderen persistenten Datei (XML oder JSON) gespeichert und abgerufen werden (siehe Abbildung 3.11). Die einzelnen Informationen zu den Services werden anschließend mithilfe des Hateoas-Controller zurückgegeben. Dieser Controller beinhaltet ähnliche Funktionen wie der in Kapitel 3.4.2 beschriebene OutputController, allerdings wurde er für die Rückgabe der Informationen der Services angepasst. Abbildung 3.12 zeigt die Anfrageverarbeitung von Prozessmodellen mithilfe des Service Discovery Providers.

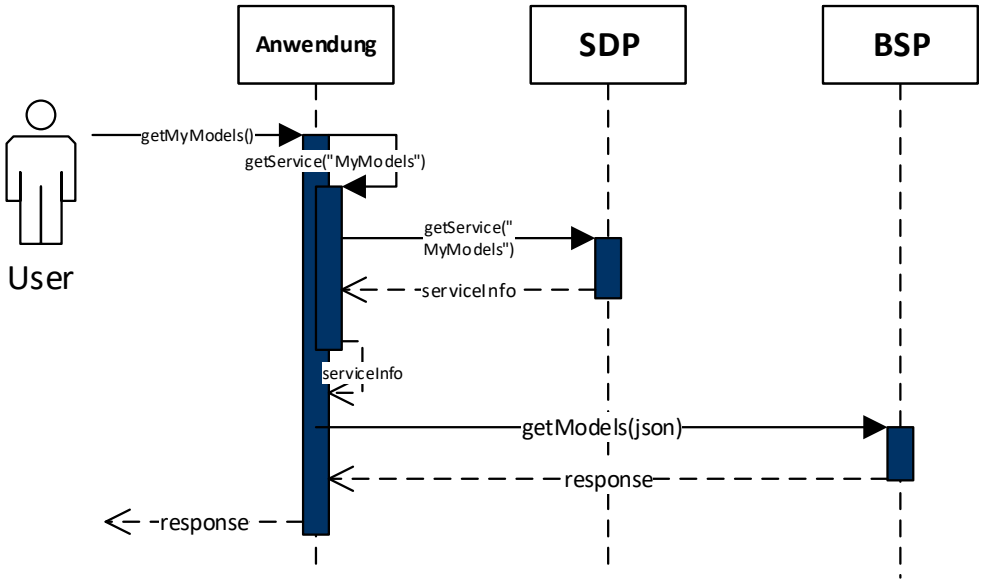

Abbildung 3.12: Sequenzdiagramm der Abfrage einer Prozessmodellliste mithilfe des Service Discovery Providers

Die Anfrage eines Schnittstellenbenutzers (siehe Abbildung 3.12) die eigenen Modelle zu lesen wird über eine Anwendung aufgerufen. Da die Anwendung vom Service nur die Bezeichnung kennt, aber nicht die URL oder weitere Parameter, wird eine Anfrage an den SDP gesendet. Der SDP verarbeitet die Anfrage und gibt Informationen des Services zurück. Anschließend kann die Anfrage an den zuständigen BSP gerichtet werden.

Routing Point Provider

Der *Routing Point Provider (RPP)* hat zwei verschiedene Aufgaben: zum Einen bietet er den Einstiegspunkt zur Schnittstelle und leitet Anfragen an den DSP oder BSP weiter. Zum Anderen bietet er Informationen über verschiedene Routen. Der Aufbau des Routing Point Providers ist in Abbildung 3.13a dargestellt und ist ähnlich dem BSP. Im RPP werden die Routen (URLs) der verschiedenen Komponenten zusammengefasst und gespeichert.

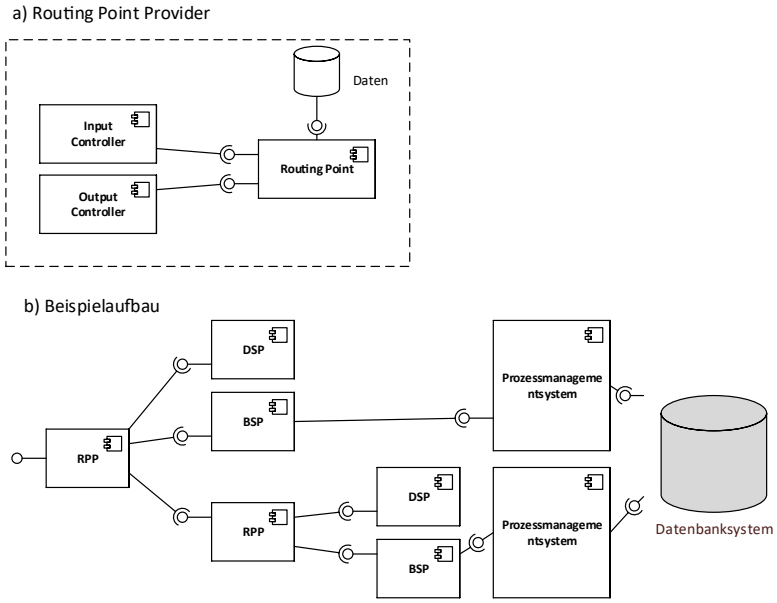

Abbildung 3.13: Komponentenmodell des Routing Point Providers

In Abbildung 3.13b ist ein Beispielaufbau mit zwei Prozessmanagementsystemen abgebildet. Hierbei verwenden beide Prozessmanagementsysteme ein gemeinsames Datenbanksystem, allerdings arbeiten die Prozessmanagementsysteme auf unterschiedlichen Servern. Eine Anfrage in diesem Aufbau kann wie in Abbildung 3.14 ablaufen. Dabei wird eine Anfrage durch die Anwendung an den ersten RPP gesendet. Dieser RPP leitet zuerst eine asynchrone Anfrage an den zweiten RPP *(RPP2)*. Dieser führt die Anfrage anschließend an den verfügbaren *SDP2* und *BSP2* weiter und antwortet dem ersten RPP. Gleichzeitig, also während der RPP die Anfrage an RPP2 sendet, wird die Anfrage über den bekannten SDP und BSP verarbeitet. Da die angeforderten Daten durch die Anfrage des RPP2 zuerst zurückgesendet wurden, wird dem BSP ein 'Abbrechen'-Nachricht gesendet und die Daten des RPP2 an die Anwendung zurückgesendet.

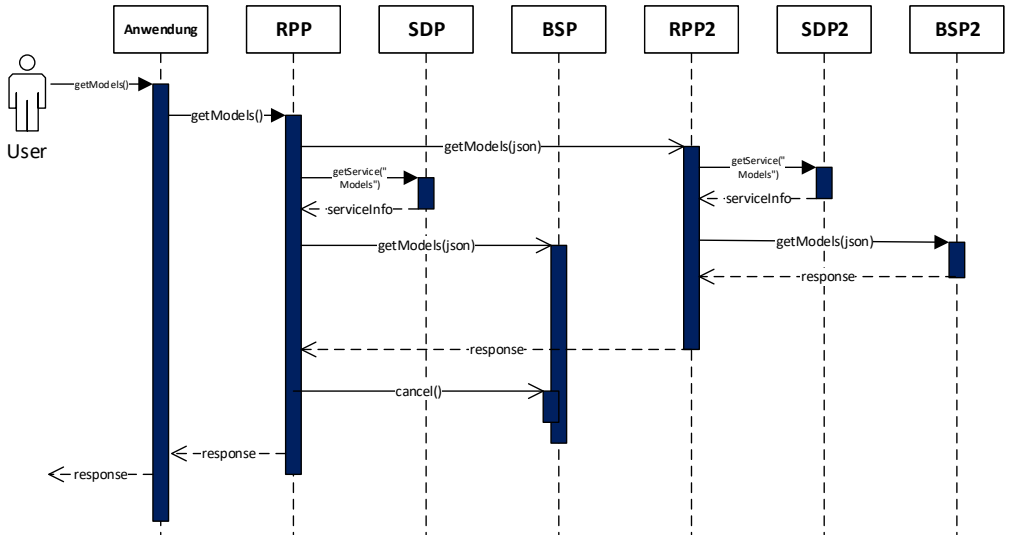

Abbildung 3.14: Sequenzdiagramm einer Anfrage mit mehreren Prozessmanagementsystemen

Anzumerken ist, dass alle Provider (Basic Service Provider, Service Discovery Provider und Routing Point Provider) auf verschiedenen Servern bzw. Instanzen ausgeführt werden können.

3.4.3 Schnittstelle zum Prozessmanagementsystem

Um die technische Anforderung *T2* (siehe Tabelle 3.3) zur einheitlichen Darstellung von Prozessmodellen, Prozessinstanzen und Prozessaktivitäten realisieren zu können ist die Schnittstelle zum Prozessmanagement sowie die Transformationen der Daten ein weiterer Aspekt der Webservice Schnittstelle.

Zunächst muss das Prozessmanagementsystem eine Schnittstelle für die Webservice Schnittstelle bieten. Dadurch können Daten gelesen und an die Anwendung zurückgesendet werden. Zusätzlich müssen Datenstrukturen, wie Prozessmodelle, Prozessinstanzen und Aktivitäten in ein allgemeines Datenmodell transformiert werden. Besitzt die Webservice Schnittstelle direkten Zugriff auf die technische Implementierung des Prozessmanagementsystems, lässt sich die Transformation durch eine *Assoziationsstrategie* realisieren (siehe Abbildung 3.15). Falls kein direkter Zugriff möglich ist und

die Datenstrukturen der Prozessmodelle, Prozessinstanzen und Aktivitäten in einem bestimmten Datenformat durch die Schnittstelle zurückgegeben werden, kann eine *Parsingstrategie* verwendet werden.

Bei der Assoziationsstrategie (siehe Abbildung 3.15b) besitzen die Datenmodelle der Webservice Schnittstelle analog einen Zugriff auf die eigentliche, technische Repräsentation der Daten im Prozessmanagementsystem. Anhand dieser Datenmodelle kann anschließend über die Schnittstellenmodelle eine einheitliche Darstellung erstellt und zurückgegeben werden. Hierzu müssen Transformation-Klassen für ein bestehendes Prozessmanagementsystem angepasst bzw. erstellt werden. Die eigentliche Logik des Webservices muss dabei nicht angepasst werden.

Die Parsingstrategie (siehe Abbildung 3.15a) wird verwendet, falls die Schnittstelle des Prozessmanagementsystem eine Repräsentation der Prozessmodelle, Prozessinstanzen oder Aktivitäten in ein generisches Datenformat (beispielsweise XML) zurückgibt. Dabei wird diese Repräsentation in ein Schnittstellenmodell konvertiert. Nach der Konvertierung können die Daten analog zur Assoziationsstrategie zurückgegeben werden.

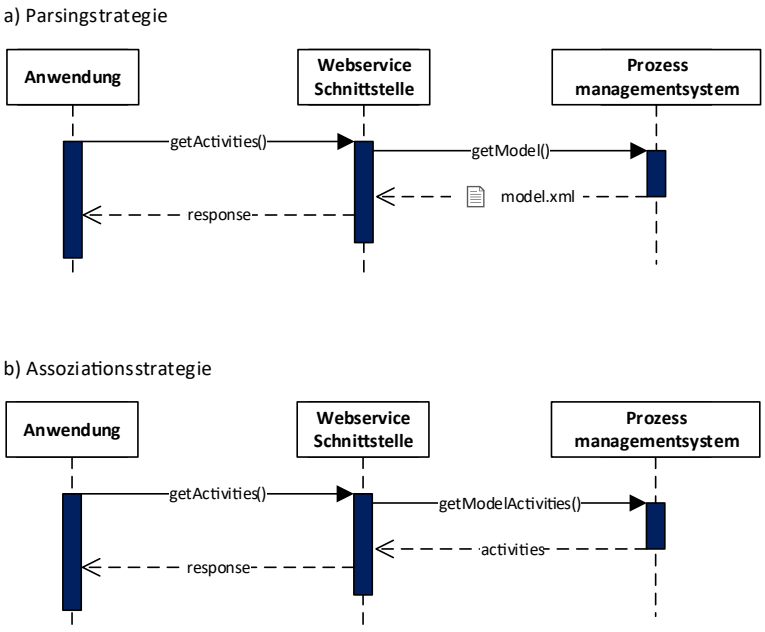

Abbildung 3.15: Sequenzdiagramm der Transformation eines Prozessmodells

3.4.4 Zusammenfassung

In diesem Kapitel wurde durch Betrachtung von drei Anwendungsfällen eine Anforderungsanalyse durchgeführt und funktionale bzw. nicht-funktionale Anforderungen definiert. Anhand dieser Anforderungen wurde ein Lösungskonzept für die Webservice Schnittstelle entworfen.

Das Konzept der Webservice Schnittstelle bietet die Möglichkeit, verschiedene Anwendungen über eine standardisierte Schnittstelle mit einem oder mehreren Prozessmanagementsystemen zu kommunizieren. Da Prozessmodelle, Prozessinstanzen und Aktivitäten eines Prozesses in eine einheitliche Darstellung transformiert werden, ist die Anbindung verschiedener Prozessmanagementsysteme einfach möglich. Zusätzlich ist keine Anpassung der den Webservice anfragenden Anwendungssoftware bei einem Wechsel des Prozessmanagementsystems notwendig.

Auf Basis des vorgestellten Lösungskonzeptes wurde eine Beispielimplementierung der Webservice Schnittstelle durchgeführt, die in Kapitel 5 beschrieben wird.

4

Clavii BPM Platform

4.1 Einführung

Die *Clavii BPM Platform* ist ein Prozessmanagementsystem, das im Rahmen mehrerer Masterarbeiten an der Universität Ulm entwickelt wurde [8]. Mit Clavii lassen sich Geschäftsprozesse leicht modellieren und implementieren, hierbei sind wenig Vorkenntnisse in BPMN und Prozessmodellierung notwendig. Zusätzlich verwendet Clavii zur einfacheren Bedienung eine Blockstruktur bei der Modellierung. Dadurch lassen sich Fehler vermeiden und die Komplexität reduzieren. Neben der einfachen Modellierung bietet Clavii eine Organisationsstruktur und ein Rechteverwaltungssystem an, mit denen sich Prozessmodelle strukturieren und die Sichtbarkeit, Bearbeitbarkeit und Ausführbarkeit auf Benutzer und Gruppen eingrenzen lässt. Clavii verwendet als Process Engine *Activiti*. Activiti ist eine Prozessmanagementplattform, die auf einer BPMN 2.0 Process Engine basiert. Durch die einfache Verwendung der Plattform in verschiedenen Java Anwendungen, auf Servern oder in einer Cloud ist es für Unternehmen ein wichtiges Tool zur Realisierung von Arbeitsabläufen und Geschäftsmodellen geworden [9].

4.2 Architekturschema

4.2.1 Clavii

Clavii ist eine BPM Plattform die für kleine Unternehmen entwickelt wurde, die nicht viel Erfahrung mit Geschäftsprozessmanagement besitzen. Clavii verkürzt die Zeit zur

Erstellung und Implementierung von Prozessmodellen. Clavii folgt den drei Designregeln *Simplicity*, *Open Standards* und *Modularization* [10].

Simplicity (Einfachheit) ist eine Verallgemeinerung und kann in *Funktionsumfang*, *Handhabung* und *Präsentation* unterteilt werden. Da die Zielgruppe von Clavii auf Mitarbeiter eines Unternehmens abzielt, die wenig oder keine Kenntnisse in BPMN oder anderen Prozessmanagementsystemen besitzen, wurde der *Funktionsumfang* der verwendeten Prozesselemente (siehe Kapitel 2.1.2) reduziert. Bei der Reduzierung der Prozesselemente wurden die Ereignisse, bis auf 'Start- und Endereignis', entfernt und die Gateways auf 'XOR- und OR-Gateways' reduziert. Damit die Prozessmodelle übersichtlich bleiben wurde bei der Verwendung von 'XOR- und OR-Gateways' auf die Block-Struktur eingehalten. Dies bedeutet, falls ein XOR-Gateway oder ein OR-Gateway eingefügt wurde dieses auch wieder geschlossen wird. In der Abbildung 2.5 wurde das 'Standard XOR-Gateway' als Block-Struktur aufgebaut. Da die Block-strukturierten Prozessmodelle leichter zu verstehen sind wurden diese in Clavii übernommen. Zusätzlich wurde das Prinzip der 'Richtigkeit während der Erstellung' hinzugefügt und erweiterte Prozessmodellkonstrukte, wie beispielsweise Events, entfernt um den Benutzer nicht zu überfordern.
Clavii ist eine Webbasierte Plattform, somit ist auf der Client-Seite keine weiteren Installationen oder Konfigurationen notwendig. Die Benutzeroberfläche sollte Funktionalitäten des BPM Lifecycle anbieten. Zusätzlich sollte jeder Schritt im BPM Lifecycle nahtlos ausführbar sein ohne umständliches Ändern des Kontext (der Oberfläche). Dadurch bietet Clavii *Einfachheit bei der Handhabung*. Die Benutzeroberfläche sollte intuitiv ohne Schulung anwendbar sein.
Open Source Frameworks wie Hibernate und Activiti wurden während der Entwicklung hinzugefügt [10].

Clavii folgt der MVC-Architektur [8, 10] und wurde als integriertes Java EE-Projekt entwickelt. Dadurch können Teile der Plattform aufgeteilt und ausgetauscht werden. Die Modularität von Clavii spiegelt sich in der Aufteilung der Modulen wider. So definiert das *Model* die am häufigsten verwendeten Objekte, wie beispielsweise Prozessmodelle, Benutzerdefinitionen oder File-Container. Die *View* beinhaltet die gesamte Logik, welche zum erstellen der Oberfläche benötigt wird, und behandelt Benutzerinteraktionen. Die gesamte Geschäftslogik wird vom *Controller* bereitgestellt. Durch das *Model* werden

Objekte wie Benutzer, Gruppen, Prozessmodelle, Anhänge oder Einstellungen definiert und durch das Hibernate Framework persistent gehalten. Die *View* stellt die Benutzeroberfläche (UI) zur Verfügung. Diese Benutzeroberfläche basiert auf Google GWT und ermöglicht den Zugriff auf alle Funktionalitätenvon Clavii [10].

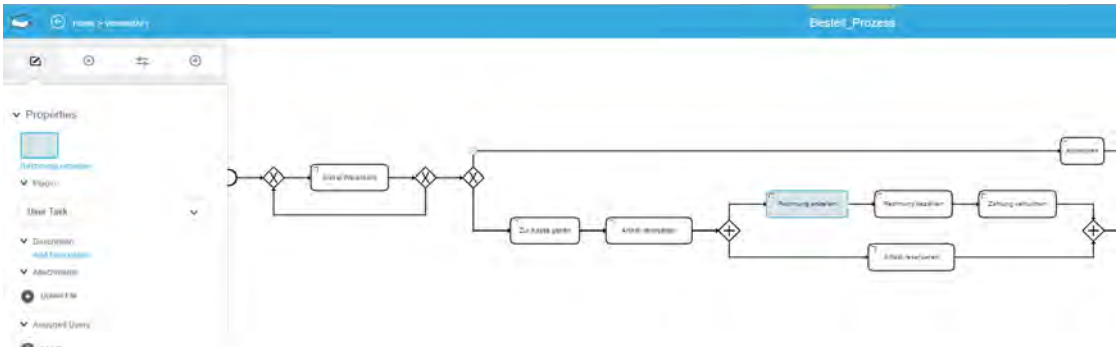

Abbildung 4.1: Clavii Oberfläche - Bearbeitung eines Prozessmodels

In Abbildung 4.1 ist die UI während der Bearbeitung eines Prozessmodells abgebildet. Hierbei befindet sich auf der rechten unteren Seite ein Ausschnitt der graphischen Darstellung des Prozessmodells aus Kapitel 2.1.2. Wird im Prozessmodelleditor eine Aktivität ausgewählt, erscheinen auf der linken Seite die Eigenschaften, welche in diesem Bereich bearbeitet und angepasst werden können.

Im *Controller* befinden sich alle serverseitigen Funktionen von Clavii. Die Architektur des Clavii Controllers werden in Abbildung 4.2 dargestellt und umfasst folgende Java-Pakete: *Identity*, *Monitoring*, *Plugin*, *Persistence*, *Run-time*, *ProcessModel* und *Validation* [10].

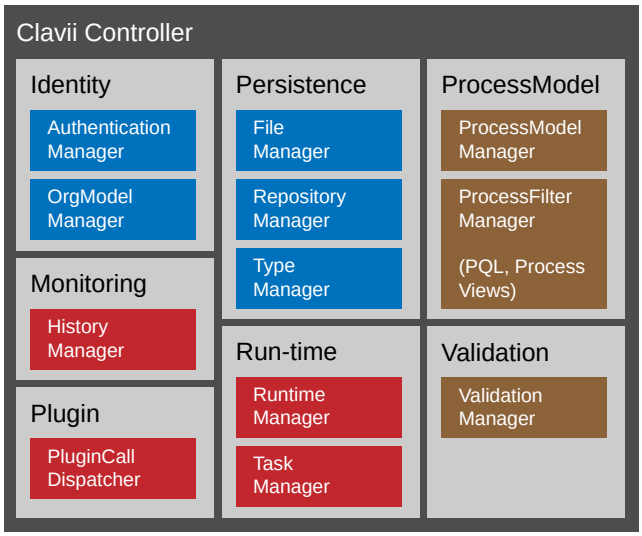

Abbildung 4.2: Clavii Architektur

Zur Identifikation und Autorisierung bietet Clavii Funktionen, die auf zwei verschiedenen Managern aufgeteilt wurden. Diese beiden Manager gehören zum **Identity** Paket. Der erste Manager ist der AuthenticationManager, dadurch werden Benutzer im System, die Agents genannt werden, autorisiert, eingeloggt, erstellt, bearbeitet oder gelöscht. Der zweite Manager ist der OrgModelManager. Dieser Manager bietet Funktionen um Organisationen, Organisationseinheiten und Rollen zu verwalten. Clavii besitzt einen eingebauten LDAP-Connector. Das **Monitoring** Paket beinhaltet den HistoryManager. Dadurch können Daten aus den ausgeführten Instanzen gelesen und überwacht werden. Durch den HistoryManager können Instanzen auf Vollständigkeit überprüft werden. Das **Persistence** Paket beinhaltet den FileManager, RepositoryManager und den TypeManager. Clavii verwendet den RepositoryService von Activiti um Prozessmodelle zu speichern und auszuführen. Die Abhandlung der persistenten Daten die zu einem Prozessmodell werden durch den RepositoryManager ausgeführt. Zusätzlich bietet der RepositoryManager Funktionen, um Prozessmodelle zu importieren, aktualisieren und zu löschen. Außerdem stehen durch den RepositoryManager Suchfunktionen zur Verfügung, um beispielsweise Prozessmodelle finden zu können, die zu einer bestimmten Organisationseinheit gehören. Die beiden implementierten Manager ProcessModelManager und

ProcessFilterManager gehören zum **ProcessModel** Paket. Der ProcessModelManager bietet Funktionen, um bestehende Prozessmodelle zu ändern, während der ProcessFilterManager Funktionen für Prozesssichten implementiert [11, 12]. Das **Run-Time** Paket beinhaltet den RuntimeManager und den TaskManager. Der RuntimeManager is für die Interaktion mit der Activiti Engine verantwortlich. Dabei triggert der RuntimeManager die Umwandlung eines Prozessmodels zu einer Prozessinstanz, welche dann von der Activiti Engine ausgeführt wird. Der TaskManager bietet Funktionen um Aufgabenlisten eines Benutzers zu lesen und ermöglicht die Ausführung von Änderungen an den Zuständen der Aufgaben. Beide Manager trennen die Clavii-Geschäftslogik von Funktionen der Activiti Engine. Der ValidationManager befindet sich im **Validation** Paket und bietet Funktionen um sicherzustellen, dass ein Prozessmodell fehlerfrei ist. Dabei wird sichergestellt, dass jeder Pfad im Prozessmodell irgendwann zum Endereignis führt und die Block-Struktur eingehalten wird. Der PluginCallDispatcher ist ein Teil des **Plugin** Pakets. Die Plugin Architektur von Clavii ermöglicht eine Erweiterung der unterschiedlichen *Tasks*, die durch Clavii ausgeführt werden können. Beispielsweise gibt es Plugins, die eine automatische Dropbox-Verbindung aufbauen und Daten dorthin senden können. Prozessdatenelemente werden nicht im Prozessmodell integriert, sondern von Clavii in einem **Data Type Framework** verwaltet [13]. Diese Datenelemente sind hierarchisch aufgebaut und durch folgende Typen definiert: SingleTypeEntity, ComplexTypeEntity und EnumTypeEntity. SingleTypeEntity sind einfache Datentypen wie String, Integer oder Boolean. ComplexTypeEntity sind Container, die mehrere verschiedene Datentypen beinhalten kann. Die EnumTypeEntity sind Datentypen die eine Auswahl angeben, wie beispielsweise eine Auswahl des Geschlechts mit den Listenelementen Männlich und Weiblich. Diese Datentypen können einfach über die Benutzeroberfläche von Clavii erweitert werden. Wie in Abbildung 4.3 zu sehen können weitere Datentypen zu einem ComplexTypeEntity durch das Betätigen des 'Add Type' Schaltfläche neue Datentypen hinzugefügt werden.

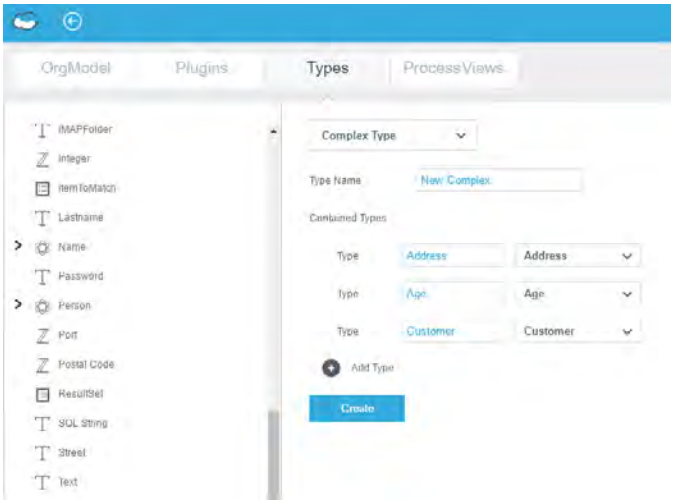

Abbildung 4.3: Clavii Oberfläche - Erstellung eines TypeEntities

4.2.2 Activiti

Die Komponenten der Activiti Platform werden in drei Kategorien unterteilt: *Modeling, Management und Runtime* (siehe Abbildung 4.4a). In der Kategorie *Modeling* befinden sich Komponenten zur Modelierung von Prozessmodellen. Die Kategorie *Modeling* beinhaltet die Komponenten *Activiti Modeler* und *Activiti Designer*. Beide dieser Komponenten werden verwendet um Prozessmodelle zu realisieren, dabei werden durch den *Activiti Modeler* die Prozessmodelle in BPMN 2.0 grafisch in einem Browser bearbeitet und erstellt, wohingegen die Komponente *Activiti Designer* ein Eclipse-Plugin ist, welche die Erstellung und Bearbeitung von Prozessmodellen in einer IDE ermöglicht [14].

Die Kategorie *Management* beinhaltet die Komponente *Activiti Explorer*. Diese Komponente ist eine Webanwendung, welches einen Zugang zur Activiti Engine bietet. Im *Activiti Explorer* können die Benutzer des Systems den Taskmanager verwenden und dadurch beispielsweise die persönlichen Aufgaben auflisten, abschließen und neue Aufgaben erstellen. Zusätzlich bietet der Taskmanager Möglichkeiten zur Auflistung aller Benutzer die zu einer bestimmten Aufgabe zugeordnet sind, Erstellung von Unteraufgaben und eine Zuweisung von Aufgaben. Neben dem Taskmanager bietet der *Activiti*

Explorer Funktionen um den Status der Activiti Engine zu überprüfen, die Prozessmodelle zu verwalten und das Gesamtsystem zu überwachen.

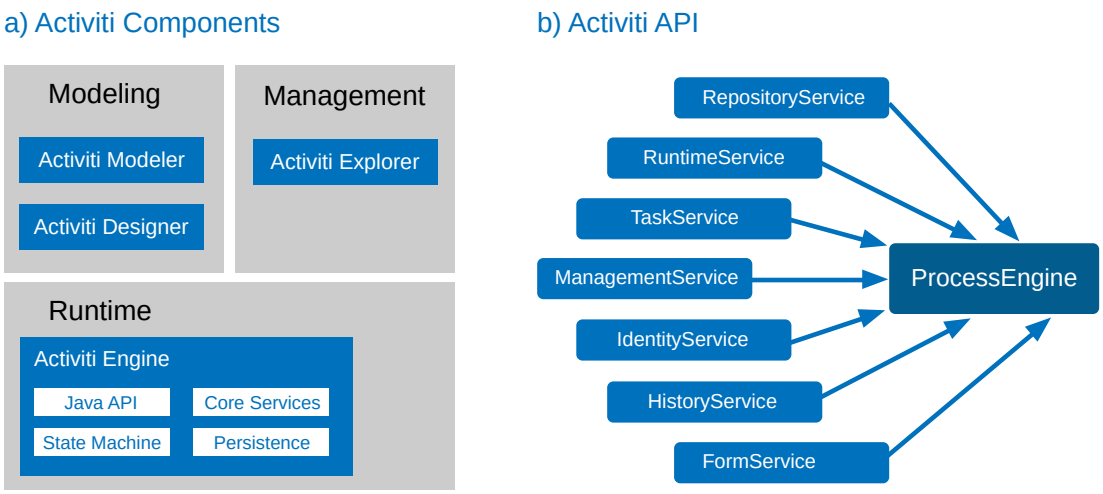

Abbildung 4.4: Activiti Komponenten

In der Kategorie *Runtime* befindet sich die Komponente *Activiti Engine*. Diese Komponente kann für die Entwicklung von Erweiterungen und der Verwendung von Activiti in anderen Java Anwendung verwendet werden. Die Activiti Engine beinhaltet eine Java API, Core Services und die Process Virtual Machine (PVM). Die PVM selbst besteht aus einer State Machine [15] und einer Persistence Ebene, welche mit der unterliegenden Datenbank kommuniziert [10].

Die Java API beinhaltet wichtige Services, die bei der Weiterentwicklung und Verwendung von Clavii verwendet werden können damit auf bestimmte Daten und Funktionen zugegriffen werden kann (siehe Abbildung 4.4b). Im folgenden werden die Hauptservices der Activiti API beschrieben.

Bei Verwendung der Activiti Engine ist der *RepositoryService* der erste Service, der aufgerufen wird. Dieser Service stellt Funktionen bereit mit denen Deployments und Process Definitionen manipuliert und verwaltet werden können. Eine Process Definition ist der Gegenpart eines BPMN 2.0 Prozessmodels in Java und repräsentiert die Struktur und das Verhalten eines Prozesses. Ein Deployment stellt ein Paket in Activity dar.

Dieser kann mehrere BPMN 2.0 Prozessmodelle (in XML) und anderer Ressourcen beinhalten. Neben der Möglichkeit der Anwendung, Verwendung und Löschung von Process Definitionen übernimmt dieser Service die Versionierung der Prozessmodelle. Mit dem *RuntimeService* können neue Prozessinstanzen gestartet werden. Eine Prozessinstanz ist die Ausführung einer Process Defintion. Jede Process Definition kann viele Prozessinstanzen gleichzeitig besitzen. Zusätzlich werden im RuntimeService Variablen und Daten zu einer Instanz gespeichert, welche häufig bei XOR-Gateways verwendet wird. Zusätzlich können bestimmte Prozessinstanzen abgefragt und bearbeitet werden. Der *TaskService* repräsentiert einen Aufgabenmanager. Durch diesen Service ist es möglich Aufgaben zu erstellen, einsehen, ausführen und abzubrechen. Zusätzlich können Aufgaben zu bestimmten Benutzern zugeordnet werden. Durch den *IdentityService* wird die Erstellung, Bearbeitung, Löschung und Befragung von Gruppen und Benutzern ermöglicht. Während der Laufzeit überprüft Activiti nicht den Benutzer. Dies bedeutet, das eine Aufgabe einem Benutzer zugeschrieben werden kann der im System nicht existiert. Dieses Verhalten ist beispielsweise für die Verwendung von LDAP wichtig. Der *FormService* bietet einen Zugang zur Activiti Form Engine um Formulare in Prozessmodellen zu definieren. Dieser Service ist zusätzlich für die Umwandlung der Formulare in HTML verantwortlich. Einer der wichtigsten Services für die Realisierung in Kapitel 5 in Activiti ist der *HistoryService*. Durch diesen Service können die gespeicherten Daten bei der Ausführung ausgelesen werden. Dabei können Informationen über ausgeführte Instanzen oder Aufgaben, wie beispielsweise die Startzeit einer Instanz, entnommen werden. Der *ManagementService* bietet Funktionen für den Zugang zur Activiti Datenbank und Datenbanktabellen. Dies ist nützlich, falls die von Activiti angeboteten Services nicht alle erforderlichen Methoden zur Verfügung stellt.

4.3 Implementierungsaspekte

Die vorgestellte Clavii BPM Plattform dient als Grundlage der im Kapitel 5 realisierten Webservice Schnittstelle. Dabei werden bestimmte Manager und Services der Clavii BPM Platform und der darunterliegenden Activiti Platform verwendet. Der *AuthenticationManager* dient als Grundlage der Authentifizierung der Webservice Schnittstelle.

Durch den RepositoryManager, RuntimeManager, TaskManager, ProcessModelManager und HistoryManager können Informationen des Prozessmanagementsystems entnommen und durch den Webservice bereitgestellt werden. Zusätzlich zu den Clavii BPM Managern werden die Activiti Services RuntimeService, TaskService und HistoryService bei der Realisierung der Webservice Schnittstelle mitverwendet.

5

Technische Implementierung

In diesem Kapitel wird eine prototypische Implementierung einer Webservice Schnittstelle nach der in Kapitel 3 beschriebenen Architektur vorgestellt. Es werden die verwendeten Technologien und Frameworks eingeführt und die Architektur der Server-Seite beschrieben und die entwickelten Komponenten betrachtet. Des Weiteren wurde ein Java Client entwickelt, der die Kommunikation mit der Schnittstelle implementiert.

5.1 Verwendete Technologien und Frameworks

Um die vorgestellte Webservice Schnittstelle technisch zu implementieren, ist ein Prozessmanagement erforderlich. Für die prototypische Umsetzung wurde Clavii verwendet, da der Quelltext frei verfügbar ist und Clavii verschiedene Manager zur Verfügung, welche eine Programmierung der einzelnen Webservice Funktionalitäten erleichtern.

Neben der Auswahl eines geeigneten Prozessmanagementsystems muss ein geeignetes Protokoll zur Übertragung der Anfragen gewählt werden. Dieses Protokoll muss den in Kapitel 3.4.1 beschriebenen Funktionen unterstützen. Anhand den Anforderungen des Protokolls sind HTTPS (HTTP) und CoAP mögliche Kandidaten. HTTPS setzt auf eine TCP Verbindung und somit auf eine fehlertolerante Übertragungsmethode [16]. Zusätzlich bietet HTTPS durch SSL-Verschlüsselung Sicherheit bei der Übertragung von Daten. Im Gegensatz zu HTTPS verwendet CoAP verbindungslose UDP-Verbindungen [17]. Dabei ist nicht immer sichergestellt, dass Anfragen und Antworten zuverlässig zugestellt werden. Deshalb bietet HTTPS auf Basis des zuverlässigen TCP ein geeignetes Protokoll zur Realisierung der Webservice Schnittstelle.

Viele Webservice Schnittstellen werden mithilfe des SOAP-Protokolls realisiert, allerdings ist SOAP komplex und konnte sich nicht überall durchsetzen. Als Alternative bieten sich REST-Schnittstellen an; diese sind einfach verwendbar und leicht verständlich. Zusätzlich existieren viele Client-Implementierungen für REST, die auf verschiedenen Plattformen und Frameworks verfügbar sind. REST (Representational State Transfer) ist ein Programmierparadigma für verteilte Systeme und wurde zur Maschine-Maschine-Kommunikation entwickelt [6]. Da die zu entwickelnde Webservice Schnittstelle Maschine-Manschine-Kommunikationen unterstützen soll und die Einbindung von REST-APIs durch geeignete Frameworks einfach zu realisieren sind wird eine REST-API implementiert.
Die Frameworks *Restlet*, *JAX-RS* und *Apache CXF* sind Beispiele für Java REST-Frameworks. Mit diesen lassen sich einfach und schnell REST-APIs schreiben und in verschiedene Implementierungen integrieren. JAX-RS (Java API for RESTful Web Services) ist eine Programmierschnittstelle der Programmiersprache Java und ist seit Edition 6 ein offizieller Teil von Java. Jax-RS verwendet Annotationen, um die Entwicklung der Schnittstelle zu vereinfachen und zu beschleunigen. Restlet ist ein OpenSource REST-API Framework, welches eine einfache Integration und schnelle Entwicklung von Webservice Schnittstellen verspricht. Das Apache CXF Framework ist ebenfalls OpenSource und unterstützt neben der Entwicklung von REST-Schnittstellen auch die Möglichkeit der Entwicklung von SOAP Schnittstellen. Zusätzlich lässt sich Apache CXF mit Spring konfigurieren und über Maven in ein Java-Projekt einbinden.
Im Rahmen der prototypischen Implementierung wird das Apache CXF Framework verwendet, da Apache CXF eine ausführliche Dokumentation besitzt und eine große Community existiert. Als Datenaustauschformat wird JSON verwendet, da es einfach zu lesen und leicht zu erstellen ist. Die Beschreibungen und Parameter der implementierten Ressourcen wurden mit SWAGGER-IO dokumentiert [18]. Dadurch ist eine bessere Übersicht über die Ressourcen verfügbar und die Funktionen können ausprobiert werden. Zusätzlich lassen sich technische Implementierungen über das Tool generieren.

5.2 Architekturschema

In diesem Abschnitt wird das realisierte Architekturschema beschrieben und dargestellt. Dabei werden die Klassen in Form von UML-Klassendiagramm abgebildet und die Funktionalität dieser Klassen und bestimmte Funktionen beschrieben. Zunächst werden die Hauptservices der Webservice Schnittstelle wie in Abbildung 5.1 beschrieben.

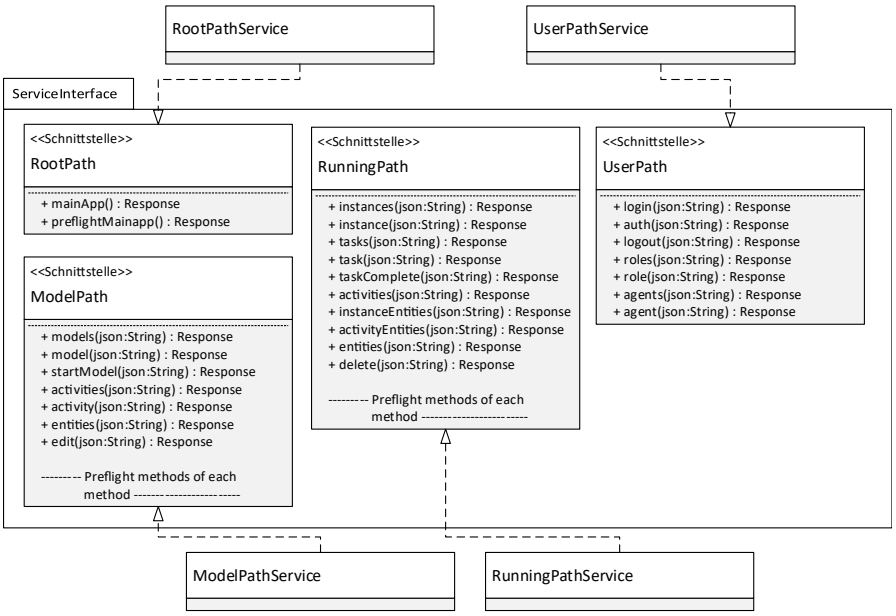

Abbildung 5.1: Klassendiagramm der Hauptservices

Abbildung 5.1 (zentral) beinhaltet alle Service Interfaces. Die definierten Funktionen der Service Interfaces müssen von den einzelnen Services implementiert werden. Die Funktion *preflightMainapp()* ist die Preflight-Methode der *mainApp()* Funktion. Eine Preflight-Mehtode muss implementiert werden. Bevor eine HTTP/1.1 Anfrage mit JSON-Daten gesendet wird, muss zuerst eine Anfrage mit der HTTP - OPTIONS - Methode abgesendet werden um zu überprüfen ob der Server (Service) Anfragen von der Anwendung entgegennimmt. Weitere Informationen können auf der Mozilla Developer Network Seite [19] nachgelesen werden.

Die Webservice Schnittstelle beinhaltet die 4 Service Klassen *RootPathService*, *UserPathService*, *RunningPathService* und *ModelPathService* (siehe Abbildung 5.1). Diese

Service Klassen lässt sich wie folgt mit der Architektur aus Kapitel 3.4.2 vergleichen: Das Paket *Discovery* wird vom *RootPathService* abgebildet, das Paket *Identity* wird vom *UserPathService* implementiert, die Pakete *TaskController* und *InstanceController* werden vom *RunningPathService* dargestellt und das *ModelController* Paket wird vom *ModelPathService* implementiert.
Das Paket der *Service Implementierung* ist in Abbildung 5.2 dargestellt. Hier sind die Klassen mit allen Attributen abgebildet.

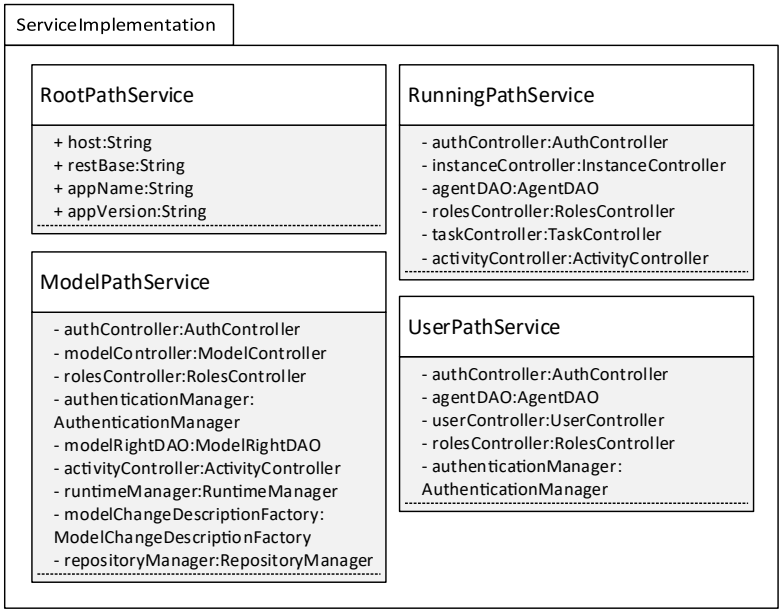

Abbildung 5.2: Klassendiagramm der Serviceimplementierung

Dabei beinhaltet beispielsweise der RootPathService in Abbildung 5.2 Attribute zur Beschreibung des Services wie beispielsweise die Bezeichnung der Applikation oder die Version. Die Attribute aus den Klassen RunningPathService, ModelPathService und UserPathService sind entweder Clavii Manager, Activiti Services oder REST-API Controller. Diese Attribute werden mithilfe von Spring injiziert und können somit leicht verwendet werden. Die angesprochenen REST-API Controller sind in Abbildung 5.3 dargestellt.

Abbildung 5.3: Klassendiagramm des Controllers

Im Paket *Controller* befinden sich Klassen, die als Controller zur Kommunikation mit Clavii und Activiti agieren. Dabei übernimmt der ActivityController die Funktionalitäten zur Erfassung der einzelner Aktivitätsdaten, der AuthController übernimmt Login und Logout sowie die Authentifizierung der Benutzer über die REST-API Schnittstelle. Der InstanceController übernimmt die Funktionalität bezüglich der Instanzen und den Inhalt der Instanzen. Durch die Funktion 'getInstanceOfUser' können mithilfe vorgegebener Parameter die zurück zu liefernden Prozessinstanzen gefiltert werden. Durch die Verwendung des UserControllers können Daten über Benutzer und Rollen entnommen werden. Damit Modelle und Informationen über Prozessmodelle aus Clavii BPM gelesen werden können, wird der ModelController über die Rest-API Schnittstelle verwendet. Der ReturnController wird für jede Rückgabe von Daten verwendet und erzeugt die benötigte 'BuildPreflight()' Funktion. Mithilfe des RolesController kann überprüft werden, ob ein bestimmter Benutzer Rechte für eine bestimmte Aktion besitzt. Der TasksController wird verwendet, um Aktivitäten und deren zugehörige Attribute zu entnehmen. Hierbei werden auch Funktionen zur Filterung dieser Daten bereitgestellt. Ein weiteres Paket ist *hateoas* (siehe 2.3.3); dieses vereint Klassen zur Erstellung von Hateoas-Links. Die Verwendung von Hateoas sollte in jeder REST-API implementiert werden, dadurch wird es den Benutzern ermöglicht durch die Webservices durchzublättern. Die Abbildung 5.4 zeigt die beinhaltenden Klassen.

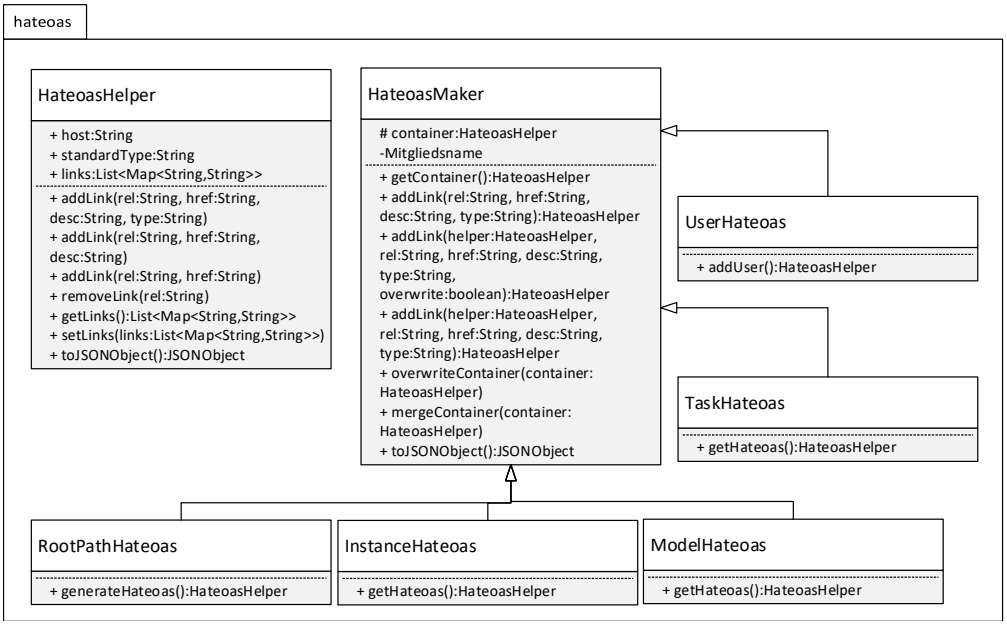

Abbildung 5.4: Klassendiagramm der Hateoas-Funktionalität

Die wichtigste Klasse innerhalb des Hateoas-Pakets ist der HateoasHelper. Dieser wird zur Erstellung der Links verwendet. Durch die Funktion 'addLink' können Links mit einem bestimmten Namen (rel), einer URL (href), einer Beschreibung (desc) und dem Typ (type) hinzugefügt werden. Die Klasse HateoasMaker ist ein Container für den HateoasHelper. Die Klassen UserHateoas, TaskHateoas, ModelHateoas, InstanceHateoas und RootPathHateoas sind Spezialisierungen der HateoasMaker Klasse und erzeugen die notwendigen Hateoas-Links für den bestimmten Bereich. Ein weiteres Paket ist das *model* Paket. Hier sind verschiedene Models abgebildet, die anschließend in ein JSON-Format umgewandelt werden. Abbildung 5.5 zeigt die Klassen des Pakets.

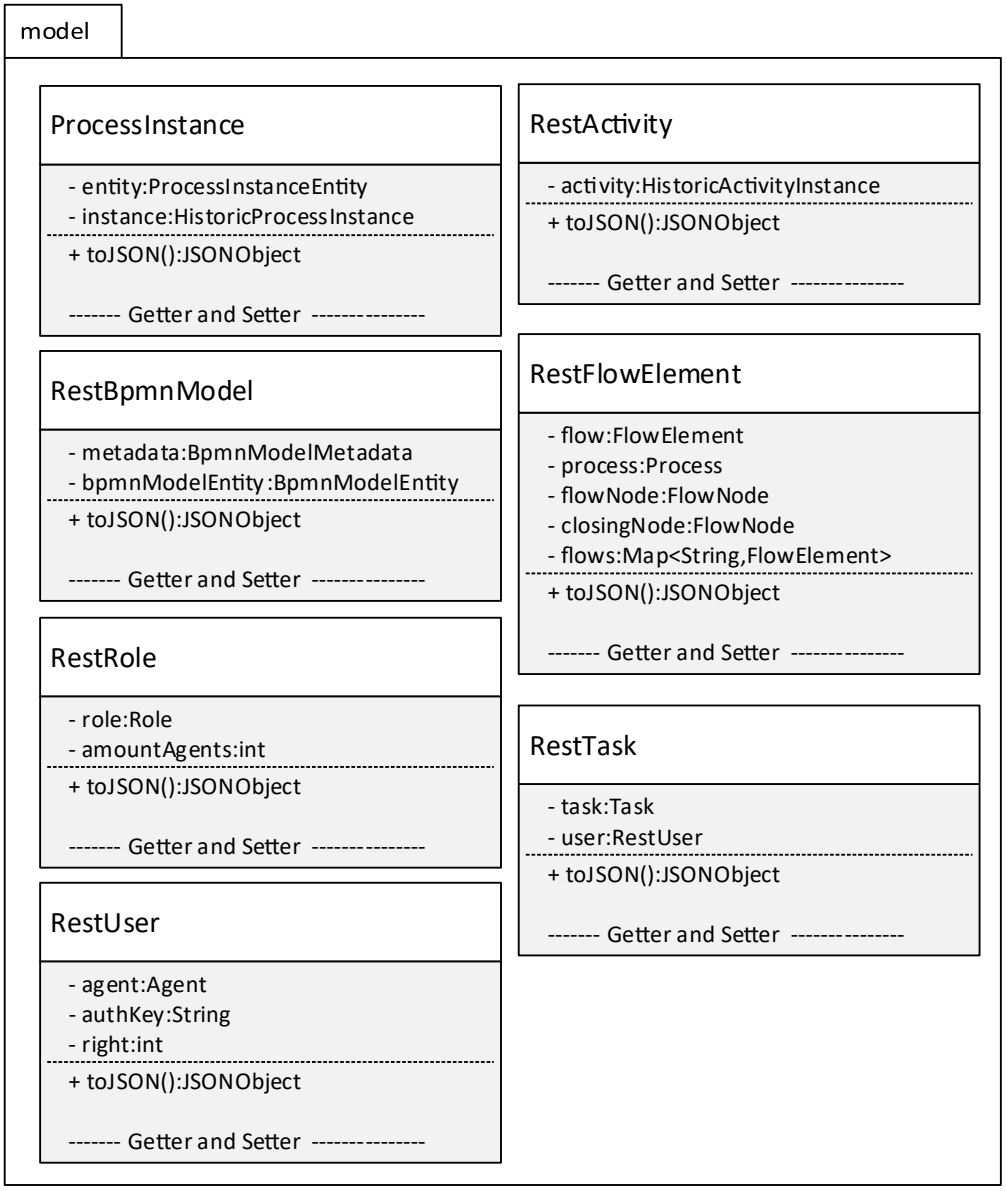

Abbildung 5.5: Klassendiagramm des Modells

Die Modell-Klassen aus Abbildung 5.5 verwenden die Model-Elemente aus Clavii BPM und Activiti. Diese Pakete bilden die Architektur der umgesetzten Webservice Schnittstelle. Im nachfolgenden Abschnitt werden die Komponenten der Architektur mit dem aus der Kapitel 3.4.2 verglichen und analysiert. Dabei werden die Unterschiede zwischen Realisierung und Konzeption beschrieben und dargestellt.

5.3 Server Komponente

Die Kategorisierung der Komponenten der Webservice Schnittstelle wurde in Abbildung 5.6 abgebildet. Dabei übernimmt der RootPathService die Komponente des Discovery Service Providers (DSP) (siehe Kapitel 3.4.2). Der RootPathService gibt Informationen über die REST-API Schnittstelle und übernimmt die Erkennbarkeit der Services. Dabei wird eine Anfrage auf die URL mit beiden Informationen gleichzeitig durchgeführt. Folgender JSON-String zeigt eine Beispielrückgabe solcher Informationen.

```
1  {
2          "appVersion": "V1.0.2",
3          "appName": "Clavii Rest API Service",
4          "baseHost": "http://localhost:8080/rapi",
5          "rest-base": "/rapi",
6          "links": {
7                  "ModelActivities": {
8                          "rel": "ModelActivities"
9                          "href": "/model/activities"
10                         "type": "POST"
11                         "desc": "This function returns all
12                         activities of an bpmnModel.
13                         expects:
14                         authKey:String [required] = Ident.
15                         of user
16                         bpmnModelId:Long[required] = bpmnModelId
17                         of bpmnModel for returning activities"
```

```
18                              },
19                      "Task": {...}, ...
20              }
21  }
```

Die URL für die Ressource *ModelActivities* setzt sich wie folgt zusammen: Die Basis-URL bietet das Attribut 'baseHost' mit dem Inhalt 'http://localhost:8080/rapi', als weiterer Teil der URL bietet das Attribut 'href' der *ModelActivities* Resource mit dem Inhalt '/model/activities'. Somit ist die URL für diese Resource 'http://localhost:8080/rapi/model/activities'. Das Attribute 'type' gibt hierbei die HTTP-Methode 'POST' an und in Attribut 'desc' sind alle nötigen Parameter aufgelistet.

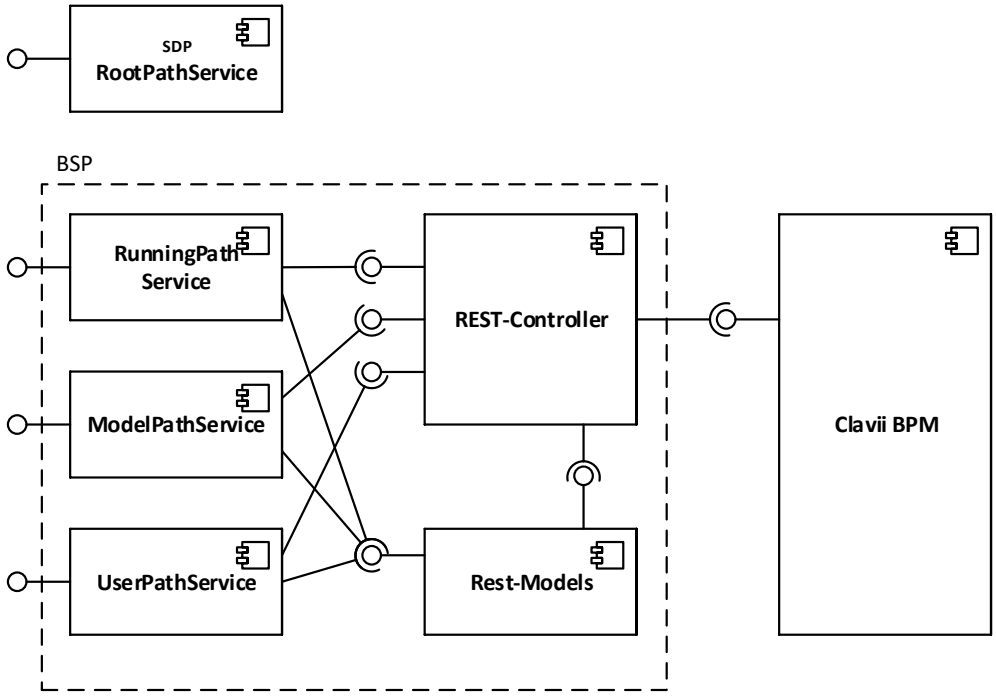

Abbildung 5.6: Komponentendiagramm der REST-API Schnittstelle

Die Services RunningPathService, ModelPathService und UserPathService sowie die Controller und Models der REST-API bilden gemeinsam den Basic Service Provider (BSP) (siehe Abbildung 5.6). Diese Services und deren Ressourcen werden durch die

vom SDP definierten URLs angesprochen. Dabei werden die Anfragen durch die Services entgegengenommen, validiert und an den richtigen REST-Controller weitergeleitet. Die Controller kommunizieren mit Clavii Managern und leiten die Ergebnisse (mithilfe der Models) an den Service zurück. Der Service sendet anschließend die Ergebnisse mithilfe des ReturnController an die anfragende Anwendung zurück. Ein Beispiel für eine Anfrage ist in der Abbildung 5.7 dargestellt.

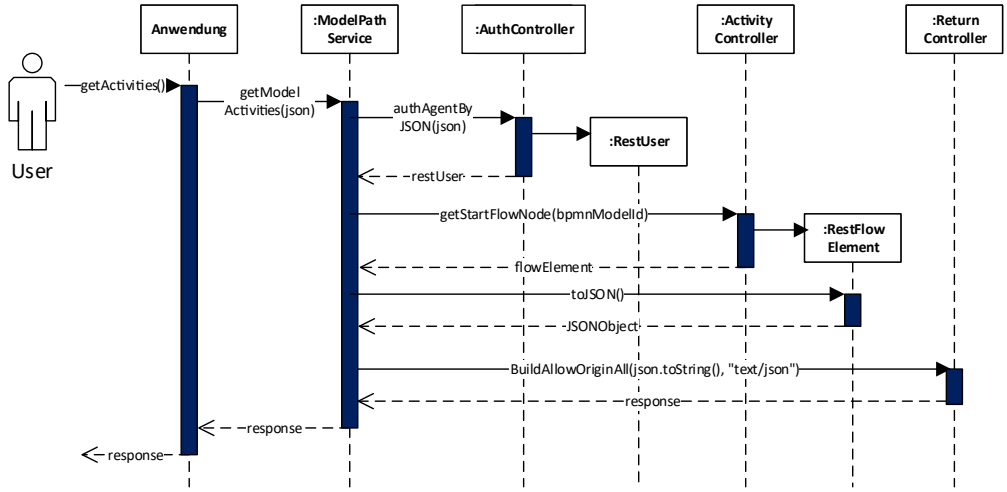

Abbildung 5.7: Abfrage der Activitäten eines Modells

Die Anfrage der Aktivitäten eines Modells werden durch den Benutzer (User) initiiert (siehe Abbildung 5.7). Dabei wird die Anfrage an den ModelPathService geleitet, dieser verwaltet die Ressource der Aktivitäten. Zur Authentifizierung wird der AuthController verwendet, dieser stellt eine Funktion zur Verfügung, die anhand eines gegebenen JSON-Strings den Benutzer authentifizieren kann. Anschließend übernimmt der Activity-Controller die Aufgabe angefragte Daten von Clavii zu lesen und in das entsprechende Datenformat zu überführen. Die Rückgabe des ActivityControlller ist ein RestFlowElement. Für die Rückgabe der Daten muss das RestFlowElement in ein JSONObject umgewandelt werden und wird abschließend im ReturnController an die Anwendung zurück gesendet.

5.3.1 Fehlerbehandlung

Da Anfragen an den Webservice auch fehlerhaft sein können oder die Rückgabe fehlerhaft bzw. leer sein kann, muss die Fehlerbehandlung durch den Webservice abgedeckt werden. Hierbei werden Fehler durch das 'status'-Attribut in der JSON-Rückgabe als Zahlencode gespeichert und als Status im HTTP-Header aufgeführt. Dabei werden folgende Statuswerte verwendet:

- *200*: Statuscode für Ok. Daten wurden erfolgreich gelesen oder die Aktion wurde erfolgreich ausgeführt
- *400*: Statuscode für fehlende Parameter. Diese Meldung erscheint wenn mindestens einer der benötigten Parameter nicht übergeben wurde.
- *401*: Statuscode für fehlende Authentifizierung oder fehlenden Rechte eine Resource mit einer bestimmten Aktion auszuführen.
- *404*: Statuscode für nicht gefunden (HTTP-Not-Found). Die angeforderte Resource konnte nicht gefunden werden oder die Anfrage ergab eine leere Liste.
- *500*: Statuscode für einen internen Fehler. Ein interner Fehler ist aufgetreten oder die Verbindung zu Clavii BPM ist fehlgeschlagen.

Zusätzlich existiert bei einem Fehlercode (außer 200) ein 'message'-Attribute. Darin wird der aufgetretene Fehler näher beschrieben.

5.3.2 Transaktionsmanagement

Da der implementierte Webservice Möglichkeiten zur Modifikation von Prozessmodellen bietet muss ein Mechanismus integriert werden um gegenseitige Transaktionen zwischen verschiedenen Anwendungen zu unterbinden und 'Lost Update' zu verhindern (siehe Kapitel 2.5). Abbildung 5.8 zeigt den Ablauf des Transaktionsmanagements mithilfe des implementierten LockController.

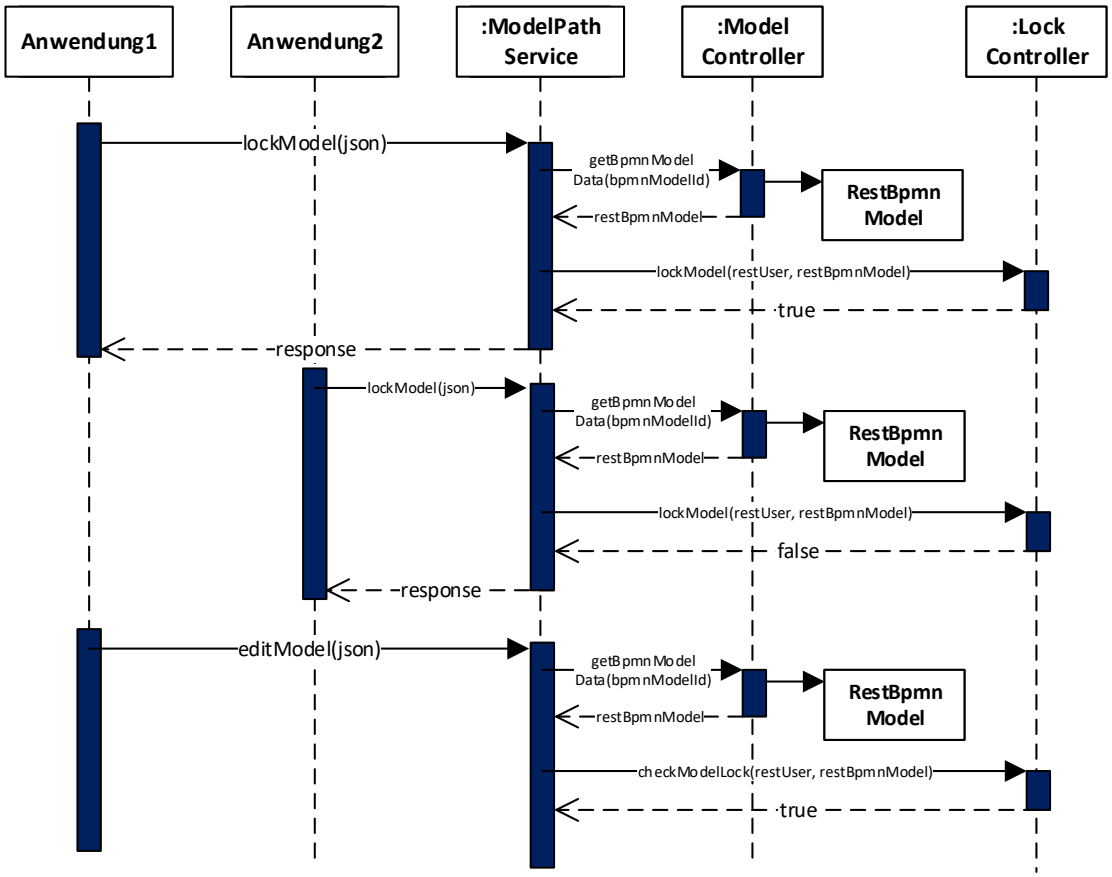

Abbildung 5.8: Transaktionsmanagement zur Verhinderung von Lost Updates

Für die Verhinderung der gegenseitigen Änderung von Modellen wird durch den Einsatz des LockController realisiert. Dabei muss ein Modell vor Änderung des Prozessmodells reserviert werden. Diese Reservierung beinhaltet einen bestimmten Benutzer, den Authentifizierungsschlüssel und eine Referenz auf das zu ändernde Prozessmodell. Da sich die 'Id' eines Prozessmodells bei jeder Version ändert, wird ein Vergleich anhand der BpmnModelMetadata-Id durchgeführt. Eine Änderungen des Prozessmodells ist ohne Reservierung nicht möglich, da vor jeder Änderung eine Abfrage zur Reservierung durchgeführt wird. Nachdem ein Prozessmodell reserviert wurde, können Änderungen an diesem vorgenommen werden. Versucht währenddessen eine andere Anwendung das Prozessmodell zu reservieren, so wird dieser Zugriff verweigert. Nachdem alle Änderungen durchgeführt wurden muss eine 'unlock' Funktion ausgeführt werden. Dadurch wird das Prozessmodell wieder freigegeben. Zusätzlich wird ein Zeitstempel der letzten Änderung bei der Reservierung gespeichert, dadurch wird bei der prototypischen Implementierung eine Reservierung nach 5 Minuten Inaktivität wieder zurückgezogen.

5.4 Java Client

Damit die entwickelte Webservice Schnittstelle verwendet und getestet werden kann wird eine Client Komponente benötigt, die mit der Webservice Schnittstelle kommuniziert. Die Komponenten sind in Abbildung 5.9 dargestellt

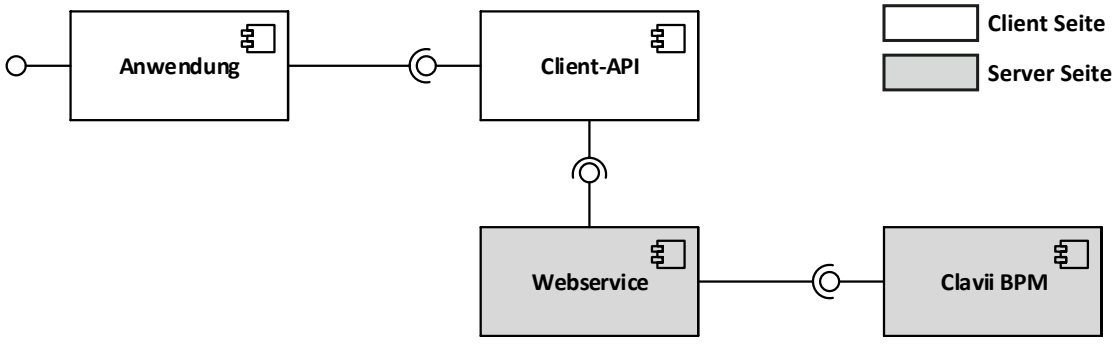

Abbildung 5.9: Komponenten der Webservice Schnittstelle mit Client-API Berücksichtigung

In Abbildung 5.9 sind diese Komponenten und deren Kommunikation untereinander abgebildet. Die Komponenten Anwendung und Client-API werden auf der Client-Seite ausgeführt, wobei sich die Webservice- und Clavii-Komponente serverseitig implementiert sind (siehe Kapitel 5.2). Die Anwendung muss dabei den genauen Aufbau der REST-Webservice-Schnittstelle nicht kennen, diese werden von der Client-API gekapselt und ausgeführt. Nachfolgend werden die Anforderungen der Client-API aufgelistet und anschließend deren Architektur betrachtet.

5.4.1 Anforderungen

Die Client-API soll in verschiedene Java-Anwendungen integrierbar sein. Zusätzlich soll eine Anwendung die Services der Webservice Schnittstelle ohne Kenntnisse über die zu sendenden Daten und URLs ausführen können. Außerdem sollen alle möglichen Services und Ressourcen durch den Java Client angesprochen und ausgeführt werden können, dies beinhaltet beispielsweise auch die oben beschriebene Filterfunktionalität.

5.4.2 Architektur

Die Architektur des Java-Clients teilt sich in den 3 Paketen *Exceptions*, *Model* und *Services* auf. Die Funktionen der Klassen innerhalb des *Services* Pakets werfen Exceptions aus dem *Exceptions* Paket und verwenden die Modelle des *Model* Paket. Zunächst werden in der Abbildung 5.13 die Klassen innerhalb des *Exceptions* Pakets dargestellt.

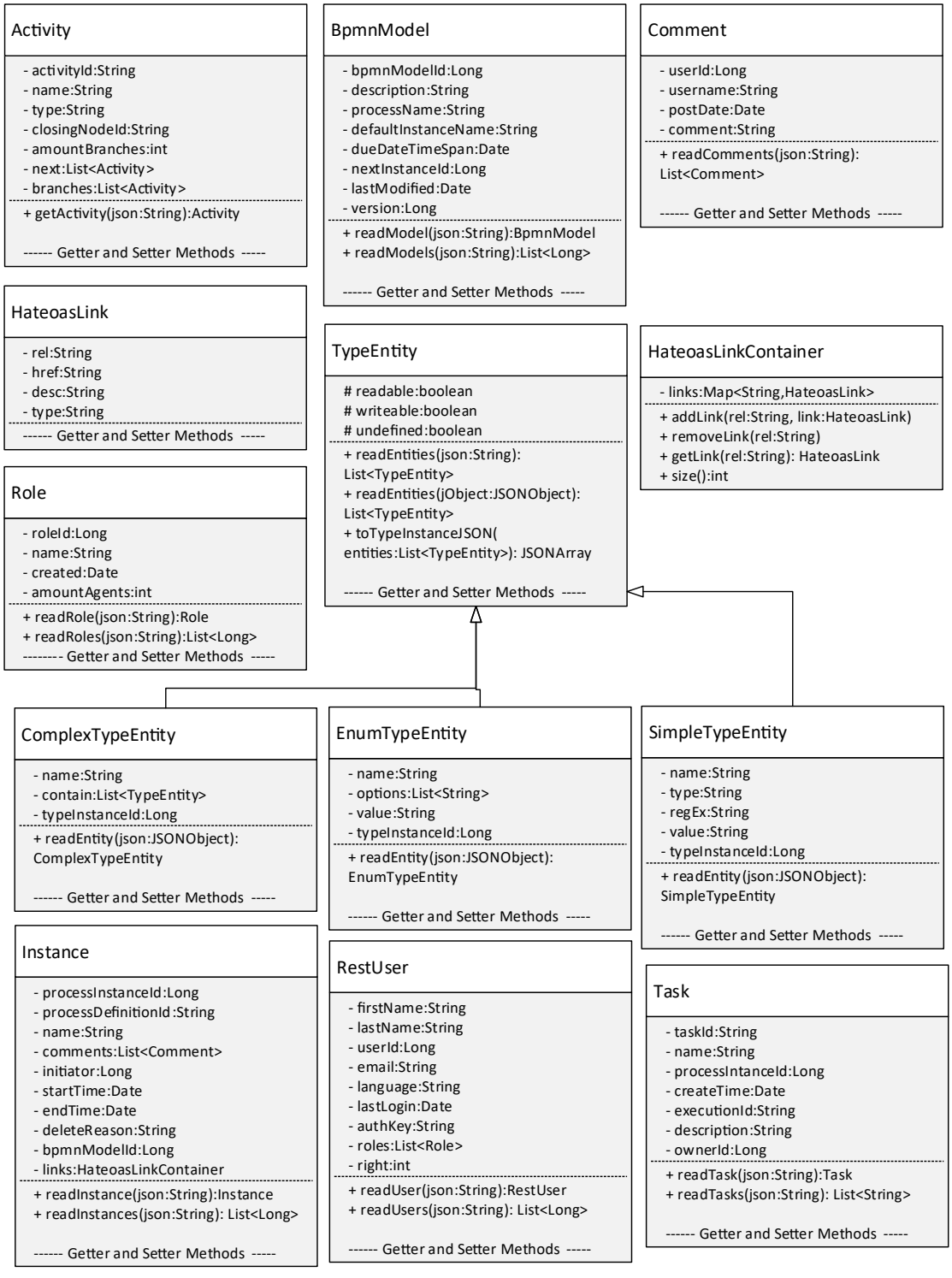

Abbildung 5.10: Java Client - Klassen des Model Pakets

Die Klassen des Pakets *Model* (siehe Abbildung 5.10) implementieren jeweils eine oder mehrere Funktionen, mit deren Hilfe ein Clavii-internes Objekt anhand eines JSON-Object erstellt werden kann. Hierbei repräsentieren die Klassen *Activity*, *BpmnModel*, *Comment*, *TypeEntity*, *Instance*, *RestUser* und *Task* die Modell-Klassen aus Clavii und der Webservice Schnittstelle. Zusätzlich bilden die Klassen *HateoasLink* und *HateoasLinkContainer* die Möglichkeit zur Speicherung der URLs der verfügbaren Services.

Abbildung 5.11 zeigt die Klassen des *Services* Pakets ab. Die Klassen innerhalb dieses Paketes kommunizieren mit der Webservice Schnittstelle.

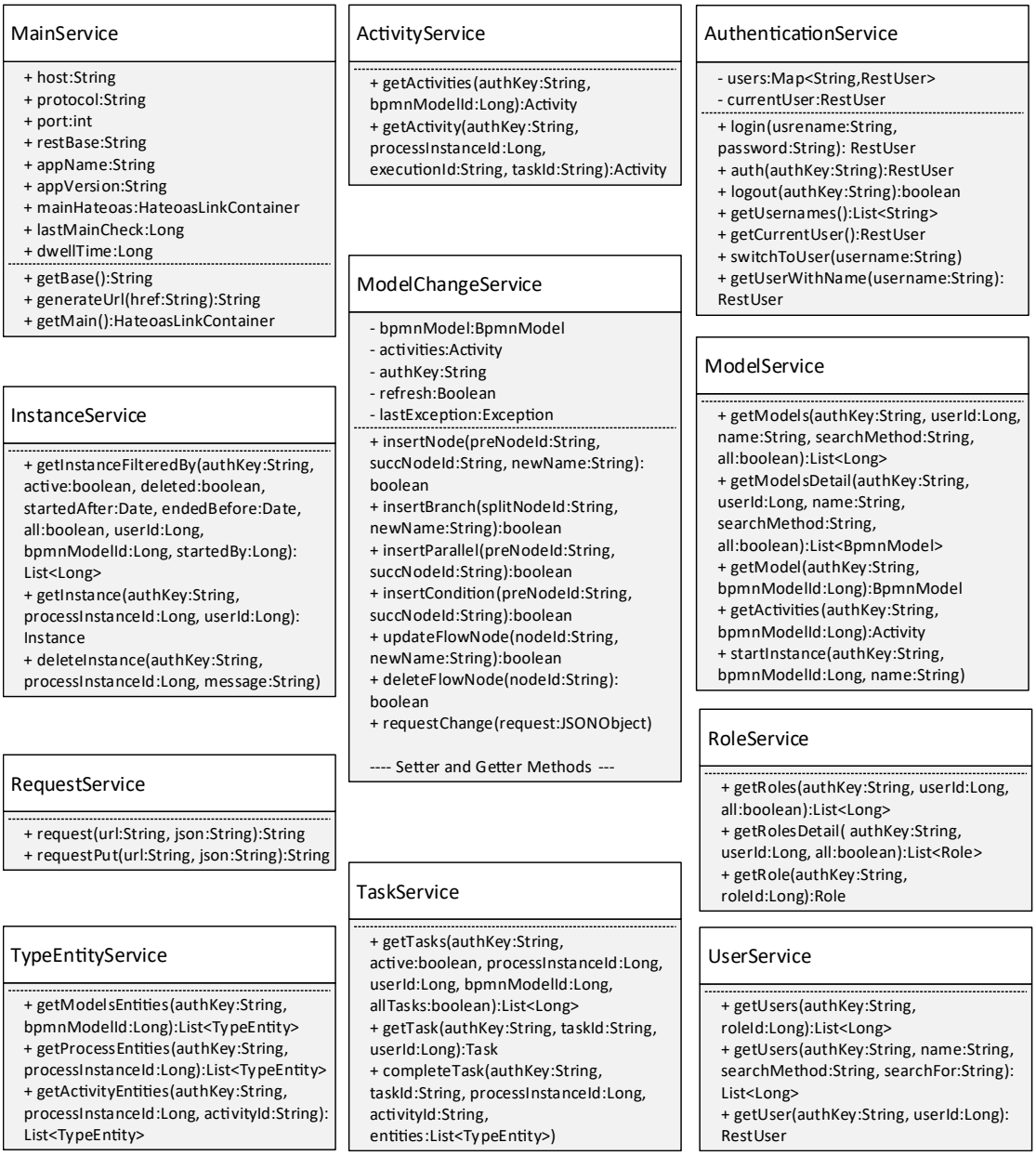

Abbildung 5.11: Java Client - Klassen des Services Pakets

Die Kommunikation zwischen dem Java Client und der Webservice Schnittstelle werden durch die Klasse *RequestService* realisiert. Diese Klasse stellt die Funktionen 'request()' und 'requestPut()' zur Verfügung. Mit der 'request()'-Funktion werden Anfragen mit der POST-Methode versendet, wohingegen durch die 'requestPut()' Funktion eine Anfrage mit der PUT-Methode gesendet werden. Der *MainService* umfasst alle Funktionalitäten, um Services der Webservice Schnittstelle zu erkennen. Um Informationen zu Aktivitäten abzufragen kann auf den *ActivityService* zugegriffen werden. Dabei können Aktivitäten von Prozessmodellen, als auch Aktivitäten von Prozessinstanzen gelesen werden. Die Authentifizierung wird mithilfe des *AuthenticationService* realisiert. Hierbei werden alle eingeloggten Benutzer gespeichert und der aktuelle eingeloggte Benutzer. Zusätzlich stehen Funktionen bereit, mit denen ein Benutzer oder eine Applikation durch einen Authentifizierungsschlüssel eingeloggt werden kann. Durch den *InstanceService* können Informationen über Prozessinstanzen gelesen werden und eine Prozessinstanz abgebrochen werden. Die Funktionen beinhalten auch alle möglichen Filterfunktionalitäten von Prozessinstanzen. Damit Prozessmodelle geändert werden können wird der *ModelChangeService* verwendet. Allgemeine Daten zu einem Prozessmodell können mithilfe des *ModelService* gelesen werden. Zusätzlich steht eine Funktion zur Verfügung, mit dem eine Prozessinstanz gestartet werden kann. Durch die Verwendung des *UserService* können Benutzerinformationen verwaltet werden. Dabei stehen Funktionen bereit, um Gruppen von Benutzern zu lesen. Diese können auch gefiltert werden. Die Anfragen zu den Rollen wird mit dem *RoleService* realisiert. Damit Aufgaben gelistet und ausgeführt werden können wurde der *TaskService* implementiert. Hier stehen Funktionen zum gefilterten Lesen der Aufgaben und der Ausführung einer Aufgabe bereit. Daten innerhalb einer Prozessinstanz, eines Prozessmodells oder einer Aktivität können durch den *TypeEntityService* gelesen werden.

Der Anfrageablauf einer Ressource, wie beispielsweise den Aktivitäten eines Prozessmodells, wird in Abbildung 5.12 dargestellt. Der beschriebene Ablauf spiegelt z.B. die Kommunikation zwischen Java Client und Webservice wider.

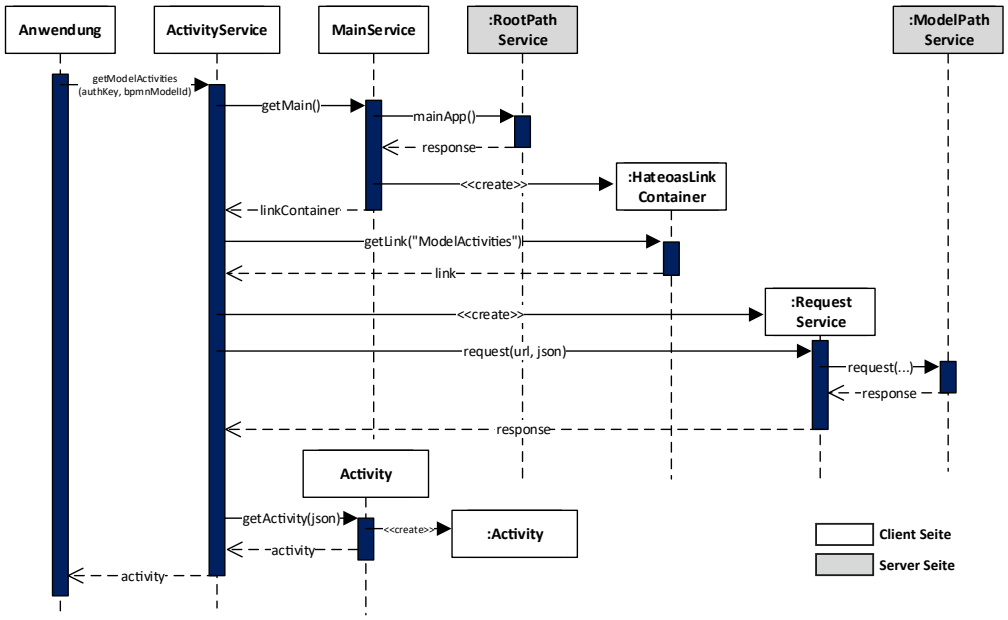

Abbildung 5.12: Anfrage der Aktivitäten eines Prozessmodells mithilfe der Java Client API

Grau markierte Objekte bzw. Klassen sind server-seitig implementiert, alle anderen Objekte und Klassen sind client-seitig implementiert. Eine Anfrage zu den Aktivitäten eines Prozessmodells werden durch die Anwendung angestoßen und an den *Activity-Service* weitergeleitet. Um den zuständigen Service zu bestimmen, wird die Funktion 'getMain()' des *MainService* aufgerufen. Dieser Service kommuniziert mit dem Root-PathService, um die Informationen zu den Services zu bekommen. Anschließend wird ein HateoasLinkContainer mit den Informationen erstellt und an den MainService zurück geleitet. Nun wird die 'getLink(...)' Funktion des HateoasLinkContainer's aufgerufen um den benötigten Link zu erstellen. Die zu sendenden Daten werden anschließend vorbereitet und mithilfe des *RequestService* eine Anfrage an den Webservice gestartet. Anschließend werden die angefragten Daten, die als Activiti-Modell vorliegen, geparsed, umgewandelt und zurück gegeben.

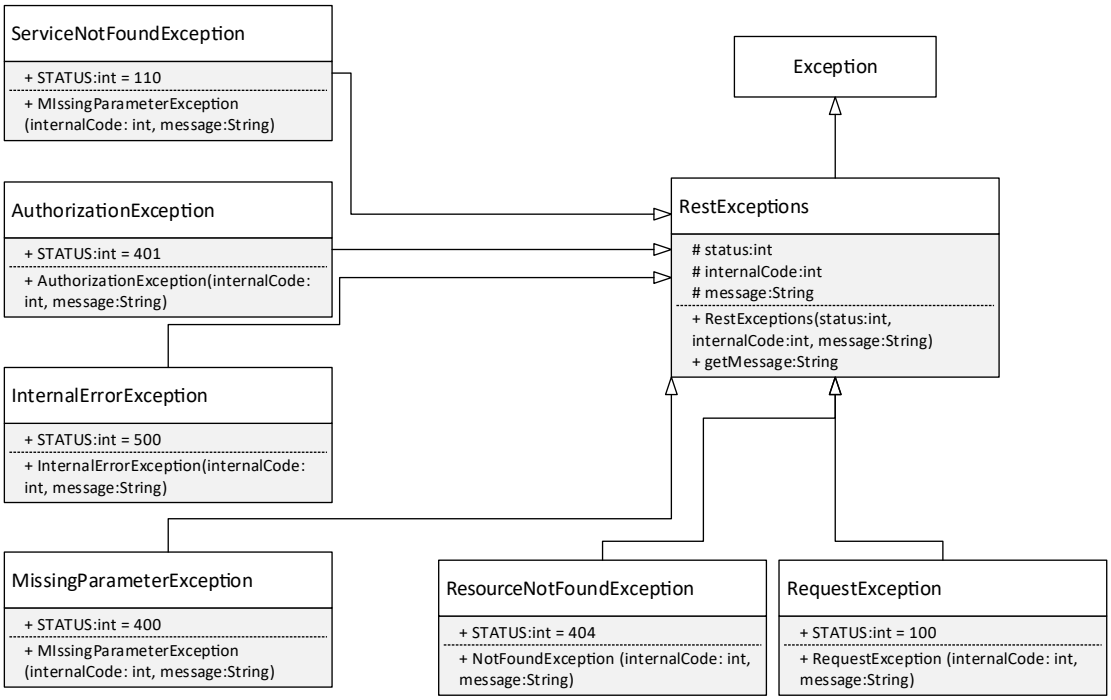

Abbildung 5.13: Klassendiagramm der Java Client Exceptions

Die Klassen innerhalb des Pakets *Exceptions* (siehe Abbildung 5.13) stellen die Exceptions die während der Arbeit mit der Webservice Schnittstelle geworfen werden können. Die Klasse *RestExceptions* ist die Oberklasse aller Exceptions und erbt selbst von der Java Exception Klasse. Diese Klasse beinhaltet den 'status' als Attribut und zusätzlich einen 'internalCode'. Der 'internalCode' selbst stellt eine interne Darstellung des Fehlers dar. Das 'message'-Attribut in der Klasse speichert die Nachricht, die bei einem Fehler mitgesendet werden. Die Klasse *AuthorizationException* ist die Ausnahme wenn ein Benutzer über die REST-API nicht authentifiziert werden konnte oder der Benutzer keine Berechtigung zur Ausführung der Aktion besitzt und implementiert den Status-Code 401. Die Klasse *InternalErrorException* wird geworfen falls die REST-API nicht angesprochen werden konnte oder durch die Schnittstelle ein Status-Code 500 zurückgesendet wird. Die Ausnahme *MissingParameterException* wird geworfen, falls mindestens ein Parameter zur Ausführung der Ressource fehlt; sie wird durch den Status-Code 400 repräsentiert. Falls eine Ressource nicht gefunden werden kann oder die Liste der zurückgelieferten Ressourcen leer ist, wird die Ausnahme *NotFoundException* geworfen. Diese verwendet den Status-Code 404. Die *RequestException* Klasse stellt die Ausnahme dar, falls eine Anfrage nicht an den Server gesendet werden kann. Dies kann geschehen falls der Server nicht verfügbar ist.

6

Verwandte Arbeiten

Dieses Kapitel analysiert verwandte Arbeiten, die einen ähnlichen Ansatz verfolgen wie diese Arbeit. Die betrachteten Arbeiten haben ebenfalls eine Webservice Schnittstelle für ein vorhandenes Prozessmanagementsystem erstellt.

6.1 IBM Business Process Manager

Der IBM Business Process Manager ist eine BPM Plattform. Er umfasst Tools für Prozessdesign, Prozessausführung und Prozessüberwachung sowie zur Optimierung von Geschäftsprozessen. Die IBM Business Process Manager Plattform bietet eine REST-API, um Daten auslesen bzw. bearbeiten zu können [20].

Im Vergleich des IBM Business Process Manager mit dem entwickelten Webservice (siehe Kapitel 3) bietet der IBM Business Process Manager nicht alle Funktionen über die REST-Schnittstelle an. Bei Betrachtung des Prozesslebenszyklus werden Funktionen der Phasen ’Prozessoptimierung’ und ’Prozessimplementierung’ (siehe Abbildung 2.2) nicht von der REST-Schnittstelle angeboten. Somit ist das Bearbeiten vorhandener Prozessmodelle und Prozessinstanzen nur durch die IBM Anwendung selbst möglich. Zusätzlich ist die Repräsentation der Prozessmodelle auf die zugrundeliegende Technik beschränkt; dadurch sind Anwendungen gezwungen die IBM-spezifische Repräsentation der Prozessmodelle zu verwenden. Der in dieser Arbeit entwickelte Webservice stellt im Gegensatz hierzu Funktionen zur Bearbeitung von Prozessmodellen und Prozessinstanzen bereit.

Die Syntax der Anfrage einer Ressource ist beim IBM Business Process Manager auf die zugrundeliegende Technik angepasst und muss von einer Anwendung implementiert werden. Im Vergleich dazu bietet die in dieser Arbeit entwickelte Webservice Schnittstelle ein generisches Datenmodell mit einem Service Discovery Provider, der den Austausch des zugrundeliegenden Prozessmanagementsystems ohne Änderung der client-seitigen Implementierung ermöglicht.

6.2 Camunda BPM

Camunda BPM ist ebenfalls eine BPM Plattform, die in Java geschrieben wurde. Mit ihr können Prozessmodell erstellt und implementiert werden. Camunda BPM bietet eine REST-Schnittstelle zur Kommunikation mit externen Anwendungen [21].

Bei Camunda BPM lassen sich nur Daten zur Prozessüberwachung über die REST-Schnittstelle entnehmen. Es werden somit keine Prozessmodellmodifikationen unterstützt. Camunda BPM fehlt die Unterstützung der Prozesslebenszyklusphasen ’Prozessoptimierung’ und ’Prozessimplementierung’ zugeordneten Funktionen (Abbildung 2.2). Zusätzlich ist die Repräsentation der Prozessmodelle und Daten, wie auch beim IBM Business Process Manager proprietär. Die in dieser Arbeit entwickelte Webservice Schnittstelle unterstützt im Gegensatz hierzu alle relevanten Funktionen, um den Prozesslebenszyklus vollständig abzudecken, und bietet zusätzlich ein generisches Format zur Darstellung von Daten und Prozessmodellen.

Die Webservice Schnittstelle der Camunda BPM ist auf das zugrundeliegende Prozessmanagementsystem zugeschnitten. Es wird eine REST-Schnittstelle angeboten, allerdings wurde auf den Support von HATEOAS verzichtet. Dadurch ist eine Verlinkung unter Ressourcen und Erkennbarkeit der Funktionalität nicht gegeben. Im Vergleich dazu bietet die entwickelte Webservice Schnittstelle ein generisches Datenmodell und einem Service Discovery Provider. Zusätzlich wird HATEOAS als Katalog für die Mächtigkeit des Prozessmanagementsystems und zur Verlinkung unter Ressourcen verwendet.

7

Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Webservice Schnittstelle für Prozessmanagementsysteme konzipiert und prototypisch implementiert. Die Webservice Schnittstelle wurde hierbei auf Basis der Clavii BPM Platform entwickelt.

Die entwickelte Webservice Schnittstelle deckt die Funktionalitäten des Prozesslebenszyklus ab und ermöglicht die Kommunikation zwischen Informationssystemen und Prozessmanagementsystemen. Dabei wurde stets auf Datensicherheit geachtet. Hierfür wurde ein generisches Request-Response Protokoll entwickelt, das eine sichere Übertragung der Anfragen sicherstellt.

Die Mächtigkeit eines Prozessmanagementsystems wird durch den sogenannten 'Service Discovery Provider' der Webservice Schnittstelle abgebildet. Dadurch können verschiedene Anwendungen verfügbare Funktionen eines Prozessmanagementsystems erkennen und auf diese einfach zugreifen. Die Verwendung von Hateoas unterstützt hierfür die Beschreibung der Funktionalitäten eines Prozessmanagementsystems. Durch ein generisches Datenmodell ist es möglich, Anwendungen und Informationssysteme auf verschiedene Prozessmanagementsysteme zugreifen zu lassen, ohne systemspezifische Darstellungsformen implementieren zu müssen.

Zukünftig sind für die Webservice Schnittstelle folgende Erweiterungen denkbar: **Abdeckung der Funktionalitäten**, **Verbesserte Unterstützung von Prozessmonitoring** sowie **Unterstützung von Live-Daten und Push-Benachrichtigungen**.

Die Webservice Schnittstelle deckt einen Teil der nötigen Funktionalität ab, zusätzlich kann die **Abdeckung der Funktionalitäten** noch erweitert werden. Die implementierte Webservice Schnittstelle kann um Funktionen wie Manipulation von Prozessinstanzen,

Migration der Prozessinstanzen zu einem geänderten Prozessmodell sowie der Unterstützung verschiedener Aktivitätstypen erweitert werden (in Clavii beispielsweise um einen *UserTask* und *ServiceTask*). Außerdem ist es sinnvoll Funktionen zur Erstellung neuer Benutzer und Zuweisung von Benutzern zu bestimmten Rollen zu implementieren.

Zur besseren Unterstützung der Prozesslebenszyklusphase 'Prozessüberwachung' kann eine **verbesserte Monitoringunterstützung** implementiert werden. So kann es zukünftig auch möglich sein bestimmte Metriken, beispielsweise die *durchschnittliche Arbeitszeit* einer Prozessinstanz für einen bestimmten Zeitraum zu lesen und zu überwachen. Dadurch können auch Dashboards besser implementiert werden. Um die Überwachung der Prozessausführung weiter zu verbessern, ist die Implementierung einer **Live-Daten und Push-Benachrichtigungsfunktion** ein weiterer möglicher Schritt. Durch eine Erweiterung um Push-Benachrichtigungen kann z.B. folgendes Szenario unterstützt werden: wird ein Prozessmodell bearbeitet, so werden alle beteiligten Benutzer des Prozessmodells benachrichtigt. Mithilfe von Live-Daten können Beendigungen von Prozessinstanzen und Aktivitäten in Echtzeit auf Dashboards angezeigt werden. Dadurch können Probleme in der Ausführung schneller erkannt und behandelt werden.

Literaturverzeichnis

[1] Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies. Springer (2012)

[2] Freund, J., Rücker, B.: Praxishandbuch BPMN 2.0. Hanser (2014)

[3] Dadam, P., Reichert, M., Rinderle-Ma, S.: Prozessmanagementsysteme. Informatik-Spektrum **34** (2011) 364–376

[4] Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: Service-oriented Architecture Best Practices. Prentice Hall (2005)

[5] Gustavo, A., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architectures and Applications (2004)

[6] Vinoski, S.: Restful Web Services Development Checklist. IEEE Internet Computing **12** (2008) 96–95

[7] Brewer, E.: Pushing the CAP: Strategies for Consistency and Availability. Computer **45** (2012) 23–29

[8] Kammerer, K., Kolb, J., Andrews, K., Bueringer, S., Meyer, B., Reichert, M.: User-centric Process Modeling and Enactment: The Clavii BPM Platform. In: Proceedings of the BPM Demo Session 2015 (BPMD 2015), co-located with the 13th Int'l Conference on Business Process Management (BPM 2015). Number 1418 in CEUR Workshop Proceedings, CEUR-WS.org (2015) 135–139

[9] Rademakers, T.: Activiti in Action: Executable Business Processes in BPMN 2.0. Manning Publications (2012)

[10] Kammerer, K.: Enabling Personalized Business Process Modeling: The Clavii BPM Platform. Master's thesis, Ulm University (2014)

[11] Kolb, J., Kammerer, K., Reichert, M.: Updatable process views for user-centered adaption of large process models. In: 10th Int'l Conference on Service Oriented Computing (ICSOC'12). Number 7636 in LNCS, Springer (2012) 484–498

[12] Kammerer, K., Kolb, J., Reichert, M.: PQL-A Descriptive Language for Querying, Abstracting and Changing Process Models. In: Int'l Conference on Enterprise, Business-Process and Information Systems Modeling, Springer (2015) 135–150

[13] Andrews, K.: Design and Development of a Run-time Object Design and Instantiation Framework for BPM Systems. Master's thesis, Ulm University (2014)

[14] Activiti Committer: Activiti Components. (`http://activiti.org/components.html`) Abgerufen: 2016-10-05.

[15] Marvin L. Minsky: Computation: Finite and Infinite Machines. Prentice-Hall (1967)

[16] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext transfer protocol–http/1.1. Technical report (1999)

[17] Shelby, Z., Hartke, K., Bormann, C.: The Constrained Application Protocol (CoAP). Technical report (2014)

[18] : Swagger API Framework. (`http://swagger.io`) Abgerufen: 2016-11-07.

[19] Mozilla Developer Network: HTTP access control (CORS). (`https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS`) Abgerufen: 2016-10-11.

[20] IBM: IBM Business Process Manager. (`http://www-03.ibm.com/software/products/de/business-process-manager-family`) Abgerufen: 2016-10-19.

[21] Camunda services GmbH: Camunda BPM Plattform - Übersicht. (`https://camunda.com/de/bpm/features/`) Abgerufen: 2016-10-19.

Abbildungsverzeichnis

Tabellenverzeichnis

Name: Andreas Alzner Matrikelnummer: 754211

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Andreas Alzner