Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Stefan Eigener

Modellgetriebene Softwareentwicklung mit BPMN 2.0 und Activiti

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Stefan Eigener

Modellgetriebene Softwareentwicklung mit BPMN 2.0 und Activiti

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Stefan Sarstedt
Zweitgutachter : Prof. Dr. Wolfgang Gerken

Abgegeben am 30.04.2013

Stefan Eigener

Thema der Arbeit

Modellgetriebene Softwareentwicklung mit BPMN 2.0 und Activiti

Stichworte

BPMN 2.0, Activiti, MDD, Java

Kurzzusammenfassung

Die manuelle Softwareentwicklung bedeutet oft, dass mit viel Aufwand erstellte Modelle vom Anfang der Entwicklung auf Dauer nicht aktuell gehalten werden und damit an Bedeutung verlieren. Diese Entwertung führt unter anderem zu erschwerten Einstiegen für neue Entwickler und zu einem fehlenden Kommunikationsmedium für fachliche Diskussionen. Im Gegensatz dazu bietet die Modelgetriebene Softwareentwicklung (auch Model Driven Software Development) einen Ansatz, der Modellen eine höhere Bedeutung zukommen lässt. Software wird bei dieser Vorgehensweise aus Modellen generiert oder ist auf andere Weise direkt an das Model gebunden. Diese Arbeit untersucht in diesem Kontext die Entwicklung von Geschäftsanwendungen auf Basis von Geschäftsprozessmodellen in BPMN 2.0. Dazu wird ein Basissystem geschaffen und mit der Activiti Process Engine verbunden. Anschließend wird die Entwicklung und das Resultat beurteilt.

Stefan Eigener

Title of the paper

Model Driven Development with BPMN 2.0 and Activiti

Keywords

BPMN 2.0, Activiti, MDD, Java

Abstract

The usual process of developing software often includes creating models with a lot of effort, which are not maintained over the whole development and / or lifecycle of the software. Because of that the models quickly lose their meaning and therefore cannot serve their purpose of helping new developers getting into the matter and being used as basis for discussions. Model Driven Development (or MDD) counters this trend by putting the model in the center of the development and, by that, also of the software. In MDD software is generated or in other wise directly related to the model. This paper examines the usage of business process models in BPMN 2.0 in a process engine (in this case of the Activiti Framework) integrated in an application and draws a conclusion, on the development itself and the result created, in the context of MDD.

Inhaltsverzeichnis

1 Einleitung 9

1.1 Gliederung 9

2 Grundlagen 10

2.1 BPMN 10

2.1.1 Definition BPMN 10

2.1.2 Elemente der BPMN 11

2.1.3 Ebenen der BPMN 14

2.1.4 Methode zur Prozessmodellierung 16

2.1.5 XML Modell 18

2.2 Activiti 18

2.2.1 Das Activiti Framework 18

2.2.2 Design Tools 18

2.2.2.1 Activiti Modeler 18

2.2.2.2 Activiti Designer 19

2.2.3 Process Engine 19

2.2.4 Activiti REST 19

2.2.5 Supporting Tools 19

2.2.5.1 Activiti Explorer 20

2.2.6 Activiti spezifische Elemente 20

2.2.6.1 Service Tasks 20

2.2.7 Spezifizierung des In-/ Output von User Tasks 21

2.2.8 Listeners 22

2.2.9 Funktionen der Activiti Engine 23
2.2.10 Repository Service 23
2.2.11 RuntimeService 23
2.2.12 TaskService 24
2.2.13 FormService 25
2.2.14 HistoryService 25
2.2.15 IdentityService 25
2.2.16 Konfiguration der Activiti Engine 26
2.2.17 Implementierungen der Activiti Engine 27
2.3 Private Krankenversicherung 28
2.3.1 Aufgaben der Privaten Krankenversicherung 28
2.3.2 Versicherungsschutz 28
2.3.3 Versicherungsvertrag 30
2.3.4 Leistungsabrechnung 31
2.4 Begriffserklärung Modellgetriebene Softwareentwicklung 32
2.5 Technische Frameworks 32
2.5.1 Spring 32
2.5.2 Maven 33
3 Fallbeispiel Leistungsabrechnung 34
3.1 Top-Level-Prozess 35
3.2 Subprozesse 37
3.3 Zuordnung von User Tasks 40
3.4 Tarife und Leistungen 40
4 Anforderungen 41
5 Konzept 44
5.1 Application Server 44
5.2 Datenbank 44
5.3 Activiti Process Engine 44
5.4 Kapselung der Engine Zugriffe 45
5.5 Webanwendung 45
5.6 Benutzer Session 45

5.7 Session Daten 45
5.8 User als Prozessvariable 46
5.9 Oberflächen 46
5.10 Servlet Klasse 46
5.11 Session Prüfung im Servlet 46
5.12 Logout 47
5.13 Fachliches Datenmodell 47
5.14 Persistierungszeitpunkt 48
5.15 Prozessabschluss 49
5.16 JPA 49
5.17 Prozessvariablen 49
5.18 User Tasks 49
5.18.1 Übergabe der Task-ID 49
5.18.2 Darstellung 49
5.18.3 Task spezifische Elemente 50
5.19 Overview 50
5.19.1 Prozessinstanzen starten 50
5.19.2 User Tasks 50
5.20 Fachlicher Komponentenschnitt 50
6 Umsetzung 51
6.1 Verwendete BPMN Elemente 51
6.2 Daten und Dokumente 53
6.3 Bewertung der Umsetzung von BPMN Elementen 53
6.3.1 Zusätzlicher Beispielprozess 53
6.3.2 Mehrfachausführung von Call Activities 54
6.3.3 Paralleler Workflow 55
6.3.4 Message Intermediate Throwing Event 56
6.4 Funktionale Erweiterungen 56
6.4.1 Kontinuierlicher Arbeitsablauf 56
6.4.2 Erweiterte GUI-Funktionalität für User Tasks 59
6.5 Fehlerbehandlung und Robustheit 61

6.5.1 Fehlerursachen und -behandlung 61
6.5.2 Prozessmigrationen 63
6.6 Performance 63
7 BPMN Elemente als Teile einer Anwendung 67
7.1 User Tasks 67
7.2 Service Tasks 69
7.3 Events 70
7.4 Bewertung 71
8 Fazit 72
8.1 Ausblick 73
Anhang A 74
Anhang B 77
Literaturverzeichnis 78
Abbildungsverzeichnis 80
Tabellenverzeichnis 81

1 Einleitung

Modellierung ist eine wichtige Aufgabe in der Softwareentwicklung. Dieser Aspekt wird jedem Student im Laufe seines Studiums beigebracht. Modelle sollen bei der Gestaltung und Planung helfen sowie zur Lebenszeit der Software diese auf grafische oder schriftliche Weise repräsentieren. Im Idealfall bilden Modelle die Grundlage für Veränderungen am System, mindestens sollten sie jedoch aktuell gehalten werden, um den dokumentarischen Wert zu behalten. In der Praxis zeigt sich jedoch oft eine Vernachlässigung der Modelle aus Kostengründen. Die Modellgetriebene Softwareentwicklung (auch Model Driven Development, kurz MDD) ist ein Ansatz, in dem das Modell einen höheren Stellenwert hat, da es mit Hilfe von Generatoren in Software umgewandelt oder das Modell durch einen Interpreter zur Laufzeit verwendet wird. Es existieren viele Modellarten, die verschiedene Aspekte aus unterschiedlichen Bereichen darstellen. Geschäftsprozessmodelle sollen dabei in dieser Arbeit verwendet werden. Sie stellen einen Modelltypen dar, der sowohl betriebswirtschaftliche als auch technische Relevanz besitzt. Da somit bereichsübergreifende Modelle entstehen, sind sie für die MDD besonders interessant, da sich die betriebswirtschaftlichen Modelle direkt auf die IT auswirken. Die Prozessmodelle sollen in die Anwendung integriert werden und den Workflow steuern.

Ziel der Arbeit ist es, MDD mit Geschäftsprozessmodellen in BPMN 2.0 anhand der Entwicklung und des Resultats zu untersuchen und zu bewerten. Zur Prozessausführung wird dabei die Process Engine des Activiti Frameworks verwendet. Die Modellierung erfolgt ebenfalls mit Tools, die in das Framework integriert sind. Die Process Engine wird dabei in eine Anwendung eingebunden, die die Ausführung von Prozessen grafisch unterstützen soll. Als Fallbeispiel für einen realen Geschäftsprozess wird die Leistungsabrechnung aus der Privaten Krankenversicherung modelliert und ausgeführt. Bewertungskriterien sind dabei die Umsetzung der Elemente gemessen am BPMN-Standard und der Aufwand für die Erstellung einer Rahmenanwendung, insbesondere der Aufwand zur Integration und Verwendung der Engine, wobei auch die BPMN selbst betrachtet werden soll und ein kleiner Vergleich zu anderen Methoden gezogen werden soll.

1.1 Gliederung

Die Arbeit ist wie folgt in mehrere Teile gegliedert:

- Grundlagen: In diesem Abschnitt werden die fachlichen und technischen Grundlagen erläutert.

    - BPMN: Es werden die Grundlagen der Notation an einem Beispiel erklärt und anschließend die Bedeutung eines Modells, sowie eine exemplarische Modellierungsmethode vorgestellt.
    - Activiti: Das Framework wird anhand seiner Bestandteile erklärt, wobei die Process Engine und ihre API im Zentrum stehen, da sie den wichtigsten Bestandteil darstellen.
    - Private Krankenversicherung: Grundlegende Eigenschaften der Privaten Krankenversicherung werden hier als Grundlage für das Fallbeispiel erläutert. Besonders hervorgehoben wird die Leistungsabrechnung.
    - Begriffserklärung Modellgetriebene Softwareentwicklung
    - Technische Frameworks: Kurze Beschreibung der in der Umsetzung verwendete technischen Frameworks
- Fallbeispiel Leistungsabrechnung: Ein fachliches Prozessmodell wird erläutert und fachliche Rahmenbedingungen, wie Beispieltarife, erläutert.
- Anforderungen: Die Anforderungen für eine Rahmenanwendung zur Ausführung von Prozessen werden als Liste aufgeführt.
- Konzept: Eine entsprechendes, die Anforderungen erfüllendes, technisches Konzept
- Umsetzung: Es werden verschiedene Aspekte, wie verwendete Elemente, Erweiterungen der Anwendung, Performance und Robustheit der Process Engine bewertet.
- BPMN Elemente als Teile von Software: Die Kernelemente der BPMN werden anhand ihrer Abbildung in Software untersucht.
- Fazit und Ausblick

2 Grundlagen

2.1 BPMN [1]

2.1.1 Definition BPMN

Die Business Process Model and Notation dient der Spezifizierung von Geschäftsprozessen. Geschäftsprozesse sind wiederkehrende Abfolgen von Handlungen, die zur Erfüllung von Geschäftszielen dienen. Neben einer grafischen Modellierung ermöglicht BPMN die Abbildung des Modells auf eine standardisierte XML-Form und macht sie damit sowohl für reine Modellierungszwecke als auch für die modellgetriebene Softwareentwicklung IT-

[1] Quelle für diesen Abschnitt ist “BPMN Method and Style” von Bruce Silver (siehe [1])

gestützter Geschäftsprozesse interessant. Dies ist möglich, da jedes Element und seine Attribute präzise semantisch spezifiziert ist. BPMN bietet die Möglichkeit sowohl parallele als auch konditionale Abläufe mit menschlichen und maschinellen Tätigkeiten zu modellieren. Zusätzlich können auftretende Ereignisse und der Umgang mit ihnen modelliert werden, außerdem kann die Kommunikation mit Entitäten oder anderen Prozessen dargestellt werden.

BPMN wurde von der OMG (Object Modelling Group) entwickelt und ist somit frei zur Verwendung. Es existieren viele Tools, sowohl Open Source als auch kommerzielle, für die Modellierung und/oder Softwareentwicklung mit BPMN.
Durch die erwähnten Eigenschaften ist BPMN die erste Modellierungssprache für Geschäftsprozesse, die gemeinsam von fachlichen als auch technischen Experten verwendet werden kann. Diese Definition der BPMN trifft ab Version 2.0 zu und im folgenden Verlauf ist mit BPMN stets diese Version gemeint.

2.1.2 Elemente der BPMN

Im vorigen Abschnitt wurde kurz auf die wesentlichen Modellierungselemente eingegangen. Tatsächlich bietet BPMN eine riesige Palette solcher Elemente. Die Erläuterung all dieser Elemente ist nicht Gegenstand und würde den Rahmen dieser Arbeit sprengen, daher sollen anhand eines Beispiels die wichtigsten Elemente erklärt werden. Weitereichende Kenntnisse können z.B. aus BPMN Method & Style von Bruce Silver [1] gewonnen werden. Die Inhalte dieses Buches bilden die Kenntnisgrundlage für den gesamten BPMN Abschnitt.
In Abbildung 1 ist ein fiktiver Beispielprozess modelliert, welcher die Erstellung eines Budgetplans abbildet. Der Prozess mit all seinen Elementen wird durch ein Pool-Element (1) begrenzt, das den Namen des Prozesses trägt. Elemente außerhalb dieses Pools stellen weitere Prozesse, Entitäten oder IT-Systeme dar, welche mit diesem Prozess kommunizieren. Die Kommunikation mit ausschließlich externen Elementen wird durch gerichtete Message-Flows (2a, 2b) modelliert, für die sowohl Quelle als auch Empfänger definiert sind. Das Brief-Symbol mit Namen zur Darstellung der Nachricht ist dabei nicht erforderlich, kann aber verwendet werden, um den Nachrichtenaustausch zu präzisieren. In diesem Fall ist das Element der Kommunikation eine Anfrage für die Erstellung eines Budgetplans durch den CFO (Chief Financial Officer) einer fiktiven Firma. Der CFO (3) ist dabei auch als Pool dargestellt. Bei externen Pools, auch wenn diese Prozesse sind, wird generell darauf verzichtet, interne Begebenheiten oder Vorgänge darzustellen. Sie werden daher als Blackbox modelliert. Man kann Prozess-Pools noch um Lanes erweitern. Diese Lanes werden einer bestimmten Rolle oder Abteilung der jeweiligen Firma zugeordnet und stellen die Aufgabenzuteilung für alle innerhalb der Lane definierten Aufgaben dar. Lanes sind, je nach Ausrichtung, horizontale oder vertikale Unterteilungen eines Pools.

Jeder Prozess hat mindestens einen Auslöser, der den Prozess startet. Diese Auslöser werden Start Events genannt. Eine Art einen neuen Prozess zu starten, ist der Erhalt einer Nachricht. Die bereits erwähnte Nachricht des CFO löst ein solches Message Start Event aus (4a). Das bedeutet, dass die Nachricht Grund des Prozessstarts ist, aber nicht, dass der Prozess zeitlich direkt nach Erhalt der Nachricht anfängt. Außerdem existieren ähnliche ereignisbasierte Start Events, wie z.B. das Timer Start Event, welches eine zeitliche Kondition an den Prozessstart knüpft. Beispiele für solche Bedingungen sind „alle 3

Monate“ oder „jedes Jahr am 1. Februar“. Allerdings kann ein Prozess auch manuell, d.h. ohne Auftreten eines definierten Ereignisses, ausgelöst werden. Grund dafür kann sein, dass ein solches nicht existiert oder die Bedingung mit den in BPMN gegebenen Elementen nicht ausdrückbar ist. Ein solches Start Event wird None Start Event (4b) genannt und bedeutet in diesem Fall, dass ein Prozess durch individuellen Bedarf von einem Mitarbeiter ausgelöst wird.

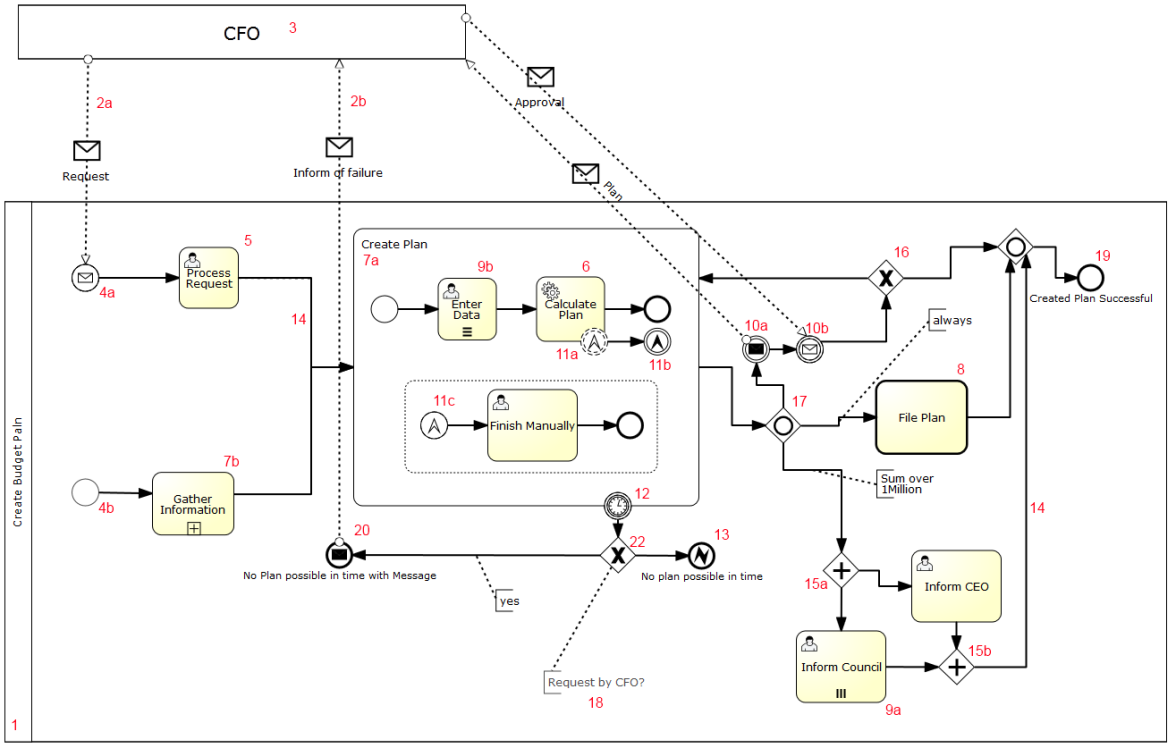


Abbildung 1 Prozessmodell zur Veranschaulichung der BPMN

Im Prozess sind verschiedene Arten von Tätigkeiten, in Folge Tasks genannt, zum Abschluss erforderlich, welche durch abgerundete Rechtecke symbolisiert werden. Generell sind alle Arten von Tasks zu benennen. Als Richtlinie dabei kann man die Tätigkeit durch ein Verb und ein Nomen beschreiben, wie im Beispiel an mehreren Stellen verdeutlicht wird. Eine Form dieser Tasks sind User Tasks. Diese bedeuten, dass eine Person an dieser Stelle im Prozess eine Tätigkeit ausführen muss. Wann diese Aufgabe abgeschlossen ist liegt im Ermessen des Bearbeiters oder weiteren Obliegenheiten, die im grafischen Modell nicht ausgedrückt werden können. User Tasks sind durch ein Personen Symbol gekennzeichnet (5). Andere Tasks, die durch Maschinen ausgeführt werden, werden Service Tasks genannt und durch ein Zahnrad Symbol gekennzeichnet (6). Solche Service Tasks können beispielsweise der Aufruf einer Java Methode oder eines Web Services sein. Auch hier kann im grafischen Modell keine genaue Definition dessen, was getan wird, erfolgen. Generell sind alle im Prozess entstehenden Daten in jedem Task verfügbar. Man kann aber auch speziell definieren und darstellen, dass bestimmte Daten (z.B. ein Dokument) Ergebnis oder Input eines Tasks sind. Tasks können auch in Form von Sub-Process-Elementen als eigene Subprozesse modelliert werden, die wiederrum Tasks enthalten. Diese Subprozesse sind Teil des Prozesses und verfügen daher implizit über dieselben Daten. Man kann sie inline (7a), also im Modell, oder hierarchical (7b), außerhalb des Modells, definieren. Beides hat dieselbe Bedeutung. Gründe für die Nutzung solcher Subprozesse können variieren, in

unserem Fall z.B. werden alle Tätigkeiten, die mit der Erstellung des eigentlichen Plans zusammenhängen als Subprozess modelliert. Subprozesse können dabei auch eigene, global definierte Prozesse sein, genannt Call Activities (8). Diese liegen nicht im selben Prozess und verfügen nur über explizit übermittelte Daten und geben entsprechend Daten zurück. Teilaufgaben, die in verschiedenen Prozessen auftreten, sollten auf diese Weise modelliert werden. Generell können Tasks und Subprozesse durch Markierung als Multiinstanz Tasks zur parallelen (9a) oder sequentiellen (9b) Mehrfachausführung gekennzeichnet werden. Im Beispiel werden dabei sequentiell Datensätze in „Enter Data“ eingegeben, die für den Plan notwendig sind, oder in „Inform Council“ parallel eine Anzahl an Räten informiert.

Auftretende Ereignisse im Prozessverlauf werden mit Intermediate Events modelliert. Mit Timer und Message Events sind bereits zwei verschiedene Eventtypen bekannt, wobei ein Event auch keine Spezifizierung des Typs haben kann. Events ohne solche Spezifizierung werden None Events genannt und stellen ein nicht genauer definiertes Ereignis dar. Die meisten Events können dabei geworfen (throwing, schwarzes Symbol) oder/und gefangen (catching, weißes Symbol) werden. Das Werfen von Events erfolgt dabei sofort, wie das Senden einer Nachricht (10a), während das Fangen einem Warten, z.B. auf eine Nachricht (10b) oder die Erfüllung einer zeitlichen Bedingung (Dauer oder Zeitpunkt), gleichkommt. Intermediate Events können dabei im Sequence Flow auftreten oder in Form von Boundary Events an Tasks und Subprocesses gebunden sein. Boundary Events können nur vom Typ „catching“ sein und zu abweichenden Sequence Flows führen, sollte im Verlauf des Tasks ein Event ausgelöst werden. Dabei wird unterschieden zwischen unterbrechenden und nicht unterbrechenden Boundary Events (gestrichelter Rahmen), je nachdem ob der normale Sequence Flow abgebrochen wird oder der normale und der Boundary Event Sequence Flow parallel verlaufen. Im Beispiel findet sich ein Timer Boundary Event (12), dass eine zeitliche Beschränkung des Tasks „Create Plan“ darstellt. Tritt die Bedingung, z.B. Dauer überschreitet 2 Wochen, ein, wird das Event ausgelöst. Boundary Events stellen die Behandlung von Events außerhalb des Tasks da, in dem sie auftreten. Mit Event Suprocesses (9) kann die Behandlung von Events innerhalb des Tasks, sollte er ein Subprocess sein, dargestellt werden. Weitere Eventtypen, die als Intermediate Events auftreten können sind u.a.:

- Error: Ein Fehler ist aufgetreten, der einen normalen Verlauf oder ein normalen Abschluss nicht möglich machen (Symbol aus 13)
- Escalation: Ein Ereigniss ist aufgetreten, welches zusätzliche Arbeit parallel oder gesondert erfordert, wie im „Create Plan“ Subprozess, in dem die maschinelle Erstellung des Plans zusätzlichen Userinput erfordert. Dazu wird dieser Bedarf in „Calculate Plan“ ermittelt und per Boundary Event (11a) verarbeitet, das ein Intermediate Event (11b) auslöst. Dieses wird im Event Subprozess (11c) behandelt.

Die folgende Tabelle enthält eine Liste wie bereits besprochene Eventtypen als Intermediate Events auftreten können. Weitere Eventtypen werden hier nicht erläutert.

Tabelle 1 Eventtabelle

Eventtyp	Im Sequence Flow	Als Boundary Event	Event Subprozess
Message	c, t	u, nu	u, nu
Timer	c	u (12), nu	u, nu
Error	-	u	u
Escalation	t (11b)	u, nu (11a)	u, nu (11c)

Aufzählung ausgewählter Eventtypen als Intermediate Events; Legende: c = catching, t = throwing, u = unterbrechend, nu = nicht unterbrechend

Aufgabenübergänge in BPMN erfolgen simultan und werden durch Sequence Flow (14) dargestellt. Ist eine Aufgabe beendet oder ein Event eingetreten wird sofort zur Folgeaufgabe oder zum Folgeevent übergegangen, wie im Beispiel oft zu sehen. Es sind aber auch Aufteilungen (branching) und Zusammenführung (merging) von mehreren Sequence Flows zu sehen, welche verschiedene Bedeutung haben. Die Tasks „Inform Council“ und „Inform CEO“ werden parallel ausgeführt. Dies ist mit einem aufteilenden Parallel Gateway (15a) gekennzeichnet. Weiterführende Aktivitäten können nur erfolgen, nachdem beide Tasks abgeschlossen sind, was durch ein zusammenführendes Gateway (15b) vorgegeben ist. An anderer Stelle ist entweder der Sequence Flow in Richtung des End Events oder eine erneute Durchführung des Tasks „Create Plan“ erforderlich. Diese Aufteilung erfolgt mit einem Exclusive Gateway (16). Im Fall des Prozessverlaufs nach Abschluss von „Create Plan“ sind je nach Bedingung verschiedene Tasks durchzuführen, allerdings sind diese nicht exklusiv verschieden und werden daher durch ein Inclusive Gateway (17) getrennt. Eine Zusammenführung der daraus resultierenden Pfade zu einem Pfad erfolgt anhand desselben Gateway-Typs vor dem Prozessende. Die Bedeutung der Zusammenführung ist dabei, dass der Gateway erst passiert werden kann, wenn alle aktiven Pfade es erreicht haben. Die Bedingungen für einen Gateway können grafisch mit Hilfe von Annotations (Beispiel 18) dargestellt werden, diese haben aber keine semantische Bedeutung.

Jeder Prozess muss mindestens ein Ende haben, und jeder seperate Sequence Flow muss in einem End Event enden. Der Beispielprozess hat drei verschiedene mögliche Enden, dargestellt durch End Events und entsprechende Bezeichnungen zur Unterscheidung. Dabei kann in diesem Fall nur jeweils ein End Event erreicht werden, was durch das Exclusive Gateway und das Interrupting Timer Boundary Event sichergestellt wird. Es ist allerdings auch möglich mehrere erreichbare End Events zu haben, wenn man Sequence Flow nicht zusammenführt. Allerdings ist dies nicht sinnvoll, da in der Regel durch diese End Events nur ein wirkliches Ende dargestellt wird, da alle aktiven Pfade enden müssen. Zudem ist ein einzelnes End Event pro logischem Ende für Betrachter des Modells verständlicher. End Events können vorher bereits besprochene Eventtypen haben. Unser Prozess hat ein None End Event (19) zur Darstellung des erfolgreichen Abschlusses. Der Erfolg des Abschlusses ist allerdings nicht an den Eventtyp gebunden, sondern durch den Titel definiert. Ein fehlerhaftes Ende wird im Prozess durch eine Message End Event (20) dargestellt. Auch hier gilt dasselbe Prinzip was den Prozesserfolg betrifft. Der Unterschied ist nur, dass im Falle dieses End Events eine Nachricht an den CFO zu schicken ist. Es ist aber auch möglich ein extra als fehlerhaft definiertes End Event zu verwenden, wie im Falle des Misserfolgs wenn kein Auftrag des CFOs vorliegt. Dadurch ergibt sich eine bessere Verständlichkeit und in unserem Falle wäre es besser auf verschiedene fehlerhafte End Events zu verzichten und nur das Error End Event (13) zu verwenden, da beide Enden prinzipiell gleich sind. Das Senden der Nachricht kann auch mit einem Message Intermidiate Event realisiert werden, welches im Anschluss in das Error End Event läuft. Der Gateway zur Fallunterscheidung ist bereits vorhanden (22).

2.1.3 Ebenen der BPMN

In Abschnitt 1 wurde erwähnt, dass die BPMN für die gemeinsame Verwendung durch fachliche und technische Experten konzipiert ist und in Abschnitt 2 wurde angedeutet wie

komplex die Sprache ist. Es ist daher sinnvoll die BPMN auf entsprechende Zwecke zugeschnitten einzuteilen. Es existieren dazu mehrere Einteilungen, z.B. nach der Workflow Management Coalition (WfMC) oder nach Bruce Silver [1, S. 6-8], die allerdings nicht Bestandteil der Spezifikation der BPMN sind. Erstere ist eine reine Einteilung der Elemente auf vier hierarchische Mengen, von denen die letzte Menge alle Elemente, die anderen aufeinander aufbauend eine eingeschränkte Menge von Elementen umfassen. Bruce Silver hingegen sieht eine Einteilung auf drei Ebenen vor. Die ersten beiden Ebenen teilen die Menge der Elemente ebenfalls in zwei hierarchische Mengen auf. Die dritte Ebene stellt eine erweiterte Sicht auf Modelle der zweiten Ebene durch das XML-Modell und die entsprechenden Konfigurationsmöglichkeiten zur automatischen Ausführung, dar.

Deskriptive BPMN [2]

Wie der Name schon sagt ist diese Ebene für den primären Zweck gedacht Prozesse mit Modellen zu beschreiben. Solche Modelle können also verwendet werden um ein gemeinsames Verständnis für die wesentlichen Abläufe und Aufgaben eines Prozesses zu erarbeiten und zu vermitteln, d.h. dass nicht zwangsweise alle Elemente des Prozesses abgebildet werden. Zu diesem Zweck ist eine Beschränkung der BPMN Elemente auf die folgende Palette vorgesehen: Pools, Lanes, User Tasks, Service Tasks, Subprozesse, None- und Timer Start Events, None-, Timer- und Cancel End Events, Parallel und Exclusive Gateways, Annotation, Message Flows und Messages. Des Weiteren sind Elemente zur Darstellung von Daten oder Datenquellen sowie technischer Link Elemente für mehrseitige Modelle vorgesehen, die in dieser Arbeit allerdings nicht verwendet werden und daher an dieser Stelle nur der Vollständigkeit halber erwähnt werden. Wie bereits erwähnt dient diese Ebene der Modellierung des Wesentlichen. Deshalb werden mit Hilfe dieser Elemente der „Happy Path“, also der einfache erfolgreiche Ablauf des Prozesses, sowie die wesentlichen, fachlich verschiedenen, fehlerhaften Verläufe modelliert. Zu dem beschriebenen Zweck kann die BPMN ohne zwangsweise korrekte oder vollständige Semantik verwendet werden. Somit ist allerdings eine Anreicherung des XML-Formats für die Verwendung in einer Process Engine für IT-gestützte Geschäftsprozesse nicht möglich und nicht vorgesehen.

Analytische BPMN [3]

Modelle, die auf der Ebene der Analytischen BPMN gemacht werden, sind als mehr als nur beschreibende Modelle gedacht. Sie sind primär für Analysezwecke im Zuge einer Prozess-Simulation vorgesehen und um den Prozess vor einer möglichen Implementierung zu testen. Solche Simulationen sind vor allem für Führungskräfte gedacht, um Prozesse anhand ihrer Resultate zu bewerten und zu verbessern. Daher sind auf dieser Ebene alle Elemente, die in der BPMN Spezifikation definiert sind, zulässig und ihrer Definition nach zu verwenden. Eine wesentliche Veränderung durch die Erweiterung der Elementpallette ist die Verwendung der im Abschnitt 1 vorgestellten Intermediate Events zur Modellierung verschiedener im Prozess auftretender Ereignisse und ihrer Behandlung. Sie ermöglichen z.B. die Darstellung weiterer im Prozess auftretender Kommunikation mit externen Entitäten oder das Senden und Warten auf Signale nach dem Publish-Subscribe-Pattern. Auch Subprozesse können auf dieser Ebene in spezieller Form gestaltet werden, wie z.B. als

[2] Siehe [1, S. 9-56]
[3] Siehe [1, S. 57-187]

Transactional Subprocess, welcher den Subprozess als einzelne Transaktion betrachtet und im Fehlerfall ein Rollback mit Compensation Boundary Events realisiert. Diese sind für jeden Task zu definieren, welcher Datensätze erstellt und sollte zu einer Rollback Routine führen. Trotz der semantischen Korrektheit und der Möglichkeit zur technischen Orientierung, wie eben durch Transactional Subprocesses angedeutet, sind Modelle dieser Ebene weiterhin nicht ohne Anreicherung des XML Modells für die Verwendung in einer Process Engine gedacht.

Ausführbare BPMN [4]

Diese Ebene ist auf die XML-Form des BPMN Modells fokussiert. Auf den beiden vorherigen Ebenen war nur die grafische Modellierung vorgesehen. Auch wenn diese Modelle durch die BPMN 2.0 automatisch eine standardisierte XML-Form haben, ist diese noch nicht für eine Verwendung in einer Process Engine geeignet. Da die Ausführung in solch einer Engine ausschließlich auf dem XML-Modell beruht, fehlen wichtige Angaben. Betrachtet man zum Beispiel ein Exclusive Gateway in einem Modell, so kann nur durch Annotations gezeigt werden, welche Bedingung geprüft wird. Diese besitzen aber keine Semantische Relevanz für den Gateway. Die eigentliche Bedingung ist nicht sichtbar und kann nur auf XML-Ebene definiert werden. Dabei bieten BPMN 2.0 Tools meist die Möglichkeit solche XML-Properties pro Element über eine GUI zu definieren. Weitere Beispiele für eine unvollständige Definition auf grafischer Ebene sind:

- Klassen / Methoden / Web Services für die Ausführung von Service Tasks
- Datenin- und output von Prozessen (Prozessvariablen) und Tasks
- Datenquellen und Fremdsysteme
- Spezielle User Interfaces mit Datenaus- und Dateneingabemöglichkeit, sowie einem Abschlussmechanismus für den Task

2.1.4 Methode zur Prozessmodellierung

Steht man vor der Aufgabe einen Prozess zu modellieren, so ist es empfehlenswert, einen Ansatz oder eine Methode zur Modellierung zu kennen und zu befolgen. Da bei nicht trivialen Prozessen meist große Modelle entstehen und Details erst beim Modellieren erkannt werden, ist ein Top-Down Ansatz, wie er von Bruce Silver [1] propagiert wird, eine Option. Im Folgenden soll diese Methode, die aus mehreren Schritten besteht, vorgestellt werden, wobei Schritte 1-5 auf Ebene Deskiptiver und Schritte 6-9 auf Ebene Analytischer BPMN erfolgen:

1) Die wesentlich Fakten klären: Was und/oder wer ist Auslöser des Prozesses? Was stellt eine jede Prozessinstanz dar? Wann und wie endet der Prozess? Dabei muss berücksichtigt werden, dass alle Aktivitäten nach dem Ende einen neuen Prozess darstellen. Ein Beispiel hierfür wäre ein Bestellprozess für Ware. Nach erfolgter Auslieferung ist der Prozess zu Ende, eine Reklamation oder Rückgabe löst einen eigenen Prozess aus.
2) Erstellen eines Top Level Diagramms für den „Happy Path“: Hierbei werden der Start und das erfolgreiche Ende als None Events, sowie alle weiteren Aktivitäten

[4] Siehe [1, S. 187-197]

modelliert. Dabei empfiehlt Silver generell die Modellierung mit Subprozessen, um die wichtigen großen Prozessschritte darzustellen. Die Subprozesse sind dabei zunächst Blackboxes. Die Modellierung sollte dabei in folgender Reihenfolge stattfinden: Pools, Lanes (optional), Start und Ende, Prozessschritte und Sequence Flow, eventuell parallele oder exklusiv konditionale Pfade unterscheiden und ins Modell einarbeiten.

3) In diesem Schritt werden die fehlerhaften Enden und ihre Auslöser identifiziert und in das Modell eingearbeitet. Folgende Unterschritte sind hier vorgesehen: Identifikation der fehlerhaften Enden und ihrer Auslöser, Wahl der End Event Typen, Einarbeitung in den „Happy Path“ mit Hilfe von Gateways.
4) Die Subprozesse werden in diesem Schritt ausgearbeitet und mit Hilfe von User und Service Tasks modelliert. Dabei ist empfohlen Subprozesse hierarchisch darzustellen und nicht inline (Beschreibung siehe 2.1.2).
5) An dieser Stelle erfolgt die Ausarbeitung der externen Kommunikation mit Message Flows. Diese Flows werden zunächst zwischen den Tasks und den externen Pools in das Modell eingezeichnet. Liegen die Tasks in einem Subprozess und hat man, wie empfohlen, die hierarchical Darstellung dieser gewählt, muss die Kommunikation auf Subprozess-Level zwischen diesem und dem externen Pool ebenfalls modelliert werden.
6) Untersuchung des bisherigen Sequence Flows und gegebenenfalls Korrekturen und Präzisierungen unter Verwendung aller in BPMN definierten Gatewaytypen.
7) Bisher wurde ein einzelnes, unspezifisches Start Event repräsentativ für alle Auslöser verwendet. In diesem Schritt werden möglicherweise verschiedene Start Kanäle identifiziert, ein Event-Typ zugeordnet und ins Modell eingearbeitet. Dies erfolgt speziell dann, wenn verschiedene Kanäle zum Beispiel eine Aufarbeitung von Daten aus einem Antrag je nach Art des Antrags voraussetzen. Sollten auf diese Weise Tasks entstehen müssen auch diese modelliert werden.
8) Bisher wurde nicht berücksichtigt, dass manche Aktivitäten generell oder unter Umständen mehrfach ausgeführt werden müssen. Solche Aktivitäten müssen identifiziert und entsprechend mit Markierung als Multiinstanz modelliert werden. Dies kann zu Änderungen in entsprechenden Subprozessen führen.
9) Bisher wurden nur fachliche Gründe identifiziert, die zu fehlerhaften Enden führen können. Dabei wurden noch keine technischen oder externen Einflüsse identifiziert, die ebenfalls zu solchen Enden oder zumindest zu einer Sonderbehandlung führen können. Dabei müssen alle möglichen Fehlerquellen identifiziert und eine Handhabung mit ihnen modelliert werden.
   Beispiel: 1)Ein Kunde sagt einen Auftrag nach Erstellung, aber vor Versand der Ware, ab. Dies wird mit Hilfe eines unterbrechenden Message Boundary Events modelliert und führt zu einem Prozessende „Bestellung abgebrochen“. 2) Ein Kunde sagt die Bestellung nach Versand der Ware ab. Dies wird mit einem nicht unterbrechenden Message Boundary Event modelliert, welches zu einem Message Intermediate Event führt, das die Mitteilung an den Kunden modelliert, dass die Bestellung nicht mehr abzubrechen ist. 3)Ein IT-System ist nicht verfügbar. Dies wird durch ein Error Boundary Event modelliert, welches auf Top Level zu einem Loop-Back führt und den Task erneut ausführen lässt.

2.1.5 XML Modell

Das XML Modell ist das eigentliche BPMN 2.0 Modell. Dabei hat das Standardmodell generell zwei Teile:

Das Prozessmodell unter dem <process>-Tag und all seine Unterelemente, wie sie von der OMG definiert wurden. Dieses Modell ist das eigentliche logische Modell und wird bei Ausführung in einer Process Engine verwendet.
XML-Namespace: *xmlns=http://www.omg.org/spec/BPMN/20100524/MODEL (Stand 27.11.2012)*

Die Verbindungselemente von den logischen Elementen und den Eigenschaften, wie Koordinaten der Eckpunkte oder Größe ihrer Darstellung, erfolgt mit <bpmndi>-Tags.
XML-Namespace: *xmlns:bpmndi=http://www.omg.org/spec/BPMN/20100524/DI (Stand 27.11.2012)*
Ein Beispiel für dieses zweiteilige Modell findet sich in Anhang A.

Die genaue Darstellung der verschiedenen Elemente ist dabei abhängig vom jeweiligen Tool und kann sich im Detail unterscheiden (z.B. die Hintergrundfarbe eines Tasks).

2.2 Activiti [5]

2.2.1 Das Activiti Framework

Activiti ist ein Java-Framework zur Modellierung und Ausführung von Geschäftsprozessen in BPMN. Es besteht zu diesem Zweck aus drei Teilen: Design Tools, Process Engine und unterstützenden / Supporting Tools. Die Teile bestehen wiederrum aus mindestens einer Komponente zur Erfüllung ihrer Aufgabe. Im Folgenden sollen diese Teile erläutert werden.

2.2.2 Design Tools

Die Design Tools dienen dem grafischen Entwurf sowie der Anreicherung dieses Entwurfs um (XML-)Parameter für die Ausführung in der Process Engine. Dabei können sowohl Attribute nach BPMN 2.0 Spezifikation als auch von Activiti Erweiterungen dieses Modells gesetzt werden. Es stehen zwei Komponenten dafür zur Verfügung.

2.2.2.1 Activiti Modeler

Der Activiti Modeler ist eine von der Firma Signavio zur Verfügung gestellte, browserbasierte Komponente zur Modellierung von Geschäftsprozessen. Sie bietet die Möglichkeit nach BPMN Spezifikation Prozesse mit Hilfe einer breiten Elementpalette grafisch zu entwerfen und die Attribute für die jeweiligen Elemente entsprechend zu setzen. Es lässt sich zum Beispiel einstellen, welcher Art ein Task ist und ob er eine

[5] Quelle für diesen Abschnitt ist „Activiti in Action“ von Tijs Rademakers (siehe [10]) und die Beschreibung des Frameworks von der Activiti-Website (siehe [13])

Multiinstanz ist mit entsprechenden Schleifenbedingungen. Das Ergebnis ist eine XML-Datei für den entsprechenden Prozess nach BPMN und eine Activiti Modeler spezifische XML-Datei für die Darstellung.

2.2.2.2 Activiti Designer

Mit dem Activiti Designer liegt ein Tool zur Modellierung von Geschäftsprozessen in Form eines Eclipse Plugins vor. Modelle im BPMN XML-Format, die mit dem Activiti Modeler entworfen wurden, lassen sich importieren und weiterbearbeiten. Es ist auch möglich Modelle komplett mit dem Designer zu erstellen. Die Elemente des Activiti Designers sind auf die Menge reduziert, die Activiti unterstützt[6]. Beim Entwurf im Modeler sollte dies dementsprechend beachtet werden. Für die Ausführung nicht relevante Elemente, wie Annotations, spielen dabei keine Rolle. Die Modelle können für die Ausführung in der Process Engine spezifiziert werden. Dabei können sowohl unterstütze XML-Attribute der BPMN Spezifikation als auch der Activiti Erweiterungen gesetzt werden.

2.2.3 Process Engine

Die Process Engine dient der Ausführung der Prozessmodelle in Activiti kompatibler Form. Damit die Modelle kompatibel sind müssen sie entsprechend mit dem Activiti Designer erstellt oder bearbeitet worden sein. Die Engine kann dabei in zwei Formen genutzt werden.

Activiti Engine
Die Activiti Engine implementiert die BPMN Spezifikation und ist die Process Engine zur Ausführung der Geschäftsprozesse. Sie ist die Kernkomponente des Frameworks und bietet u.a. die Möglichkeiten neue Prozessinstanzen zu starten und Tasks auszuführen. Im Grunde funktioniert die Engine wie ein Zustandsautomat, der die BPMN Elemente einzeln ausführt. Dabei werden Elemente wie Gateways, Events und Service Tasks automatisch ausgeführt, während der Abschluss von User Tasks manuell erfolgen muss. Neben APIs zur Erfüllung dieser Grundaufgaben bietet sie unter anderem APIs zur Historisierung von Prozessinstanzen, Identitäten-Management und Verwaltung von Prozessdefinitionen und ihren Versionen.

2.2.4 Activiti REST

Activiti REST bietet die Möglichkeit die Activiti Engine als eigene Web Application mit REST API zu verwenden. Die REST API stellt die Dienste der Engine mit Methoden, wie zum Beispiel GET oder POST, der http-Spezifikation zur Verfügung.

2.2.5 Supporting Tools

Die bisher genannten Teile des Frameworks bieten die Möglichkeit Prozesse Activiti kompatibel zu modellieren und Anwendungen zu schreiben, die diese Prozesse mit der Activiti Engine auszuführen und weitere Dienste der Engine in diesem Zuge zu nutzen. Die

[6] Eine Auflistung dieser Elemente findet sich auf der Activiti-Website [13]

Supporting Tools sollen dabei unterstützen, Prozesse und Funktionalität der Engine zu nutzen und zu testen. Dabei existiert mit dem Activiti Explorer eine Komponente, die dies ermöglicht.

2.2.5.1 Activiti Explorer

Der Activiti Explorer ist eine fertige Webapplikation mit eigener Datenbank, um die Möglichkeiten der Activiti Engine zu veranschaulichen und Prozesse zu testen. Sie kann mit einem Ant-Skript installiert und in einem Servlet Container oder Application Server deployt werden. Sie verfügt über vorgefertigte Benutzer und Prozesse und bietet u.a. folgende Funktionalität über ein Webinterface:

- Deployment von Prozessdefinitionen, Verwaltung der Deployments und Versionsverwaltung für Prozessdefinitionen
- Starten von Prozessinstanzen
- Ausführen von User Tasks
- Zuordnung von User Tasks an Benutzer
- Ermittlung des Prozesszustands

2.2.6 Activiti spezifische Elemente

Activiti bietet neben der regulären Implementierung der BPMN spezifische Attribute und Elemente als Erweiterung der BPMN zur Realisierung gleicher oder erweiternder Funktionalität an. Bei gleicher Funktionalität, wie etwa der Nutzung von Web Services in Service Tasks, können sowohl die BPMN konforme als auch eine Activiti spezifische Version genutzt werden. In anderen Fällen, wie der Definition von Methoden oder Klassen für Service Tasks, ist nur eine Activiti spezifische Version zulässig.

2.2.6.1 Service Tasks

Bei der Implementierung von Service Tasks existieren drei Möglichkeiten. Die erste und einfachste Möglichkeit ist es mit dem Attribut „activiti:class“ im Service Task eine Klasse zu referenzieren, welche das Interface JavaDelegate aus dem Paket org.activiti.engine.delegate implementiert. Dieses Interface enthält eine Methode der Signatur:
void execute(DelegateExecution execution) throws Exception.
Diese Methode wird bei Ausführung des Service Tasks von der Engine aufgerufen und stellt damit den eigentlichen Task dar. Die DelegateExecution, aus demselben Paket, wird dabei automatisch mit übergeben und stellt den Ausführungskontext im Prozess dar und kann daher unter anderem verwendet werden, um Prozessvariablen zu lesen und zu schreiben.

Beispiel:
```
<serviceTask id="serviceTask1" name="Validate order"
activiti:class="org.myProcess.myServiceTasks.ValidateService"/>
```

Klassen dieser Art können auch über das Attribut „activiti:delegateExpression“ als Service Implementierung definiert werden. Dabei muss der Wert des Attributs der Name einer

bekannten Bean sein, z.B. einer Spring- oder CDI-Named-Bean. Diese Funktionalität ist nur dann möglich, wenn die Activiti Engine Implementierung entsprechend gewählt wird (Siehe Abschnitt 2.2.17).

Beispiel:
```
<serviceTask id="serviceTask1" name="Validate order"
activiti:delegateExpression="validateServiceBean"/>
```

Unter Verwendung solcher Beans und dem Attribut „activiti:expression“ können auch Service Implementierungen genutzt werden, die nicht Implementierungen des JavaDelegate-Interfaces sind. Dies kann nützlich sein, um Methoden zu nutzen, die einen Rückgabewert haben, der in Folge als Prozessvariable genutzt werden kann. Dabei kann die Parametrisierung mit Hilfe aller Prozessvariablen und der bereits genannten DelegateExecution erfolgen. Im „expression“-Attribut können alle Prozessvariablen, referenziert über ihren Namen, verwendet werden. Die DelegateExecution verhält sich dabei wie eine Prozessvariable mit dem Namen „execution“.

Beispiel:
```
<serviceTask id="serviceTask1" name="Validate order"
activiti:expression="#{order.validate(execution)}"/>
```
Sonderfall:
`activiti:expression="#{order.odernumber}` (Stellt den Zugriff auf Objektvariablen durch einen Getter dar, in diesem Fall *„getOrdernumber()“*)

2.2.7 Spezifizierung des In-/ Output von User Tasks

User Tasks sind generell Aufgaben im Prozess die durch eine Person erfüllt werden müssen. Dabei ist im Prozessmodell nicht definierbar, neben einer Beschreibung, was genau die Aufgabe ist. Bei der Entwicklung von Anwendungen mit ausführbaren Geschäftsprozessen werden User Tasks dabei meist durch grafische Schnittstellen dargestellt. Sie geben Prozessinformationen an den User weiter und bieten die Möglichkeit für den User Input zu tätigen. Activiti unterstützt die Umsetzung solcher Schnittstellen mit Hilfe von Form Properties. Form Properties können mit Hilfe diverser Attribute konfiguriert werden, so dass sie bestimmte Zwecke erfüllen können. Dies soll an zwei verschiedenen Konfigurationen demonstriert werden, die Gesamtheit der Attribute kann der Activiti Spezifikation entnommen werden.

- Die Verwendung von Form Properties als Eingabefelder. Hierfür werden die Attribute „id“ und „name“ gefüllt. Optional kann man im Typ-Attribut angeben, welchen Typ die Form Property hat und entsprechend ein GUI-Element nutzen oder eine Wertprüfung implementieren. Ist ein Feld Pflichtfeld, so wird das Attribut „required“ auf „true“ gesetzt. Es ist möglich über das „Form Value“-Attribut Auswahlmöglichkeiten vorzugeben.
- Die Verwendung von Form Properties als Informationsträger für den Wert von Prozessvariablen. Dazu sollte man, wie im vorherigen Beispiel, „id“ und „name“ füllen. Des Weiteren muss das Feld „Variable“ mit dem Bezeichner einer Prozessvariablen gefüllt werden. Die Überschreibbarkeit kann mit dem Attribut „writeable“ festgelegt werden. Die Activiti Engine füllt das „value“-Attribut automatisch beim Lesen der Properties, wenn eine entsprechende Variable gesetzt ist.

Die Form Properties für einen User Task können mit der Activiti Engine API gelesen und Tasks entsprechend abgeschlossen werden. Form Properties können dabei in Listenform an die Engine übergeben werden. Mit Hilfe dieser Properties können grafische Schnittstellen mit Programmen erstellt oder generiert werden.
Form Properties können, wie beschrieben, auch für Start Events definiert werden.

Beispiel:

```
<userTask id="usertask1" name="User Task">
      <extensionElements>
        <activiti:formProperty id="acessedBy" name="Accessed By"
type="string"
         required="true"></activiti:formProperty>
        <activiti:formProperty id="user" name="User Id" variable="user"
         writable="false"></activiti:formProperty>
      </extensionElements>
</userTask>
```

2.2.8 Listeners

Listeners bieten die Möglichkeit auf das Eintreten bestimmter Zustände im Prozess zu reagieren, die automatisch ausgeführt werden. Entsprechend können mit Hilfe dieser Listeners Logik (technisch oder fachlich) an entsprechenden Stellen ausgeführt werden. Dabei existieren zwei Typen solcher Listeners: Execution und Task Listener. Die Zustände sind für die jeweiligen Listeners festgelegt und in der Engine implementiert (siehe Tabelle 2).

Tabelle 2 Activiti Listener

Listener	BPMN Element	Zustand	Beschreibung
Execution	Process	start	Ausgelöst durch jedes beliebige Start Event
		end	Ausgelöst durch jedes beliebige End Event
Execution	Beliebiger Task	start	Ausgelöst bei Start der Ausführung
		end	Ausgelöst bei Ende der Ausführung
Execution	Sequence Flow	take	Bei Beschreiten ausgelöst
Task	User Task	create	Bei Erstellung des User Task
		assignment	Bei Zuweisung an einen Bearbeiter
		complete	Bei Abschluss durch Bearbeiter oder Eigentümer

Die entsprechenden Listeners können dabei mit dem Activiti Designer oder im XML-Modell im jeweiligen Element definiert werden. Die Verknüpfung mit einer entsprechenden Methode oder Klasse erfolgt dabei, wie bei Service Tasks, mit „class“-, „expression“- und „delegateExpression“-Attributen. Im XML-Modell stellen diese Listener Extension-Elemente (siehe Beispiel) dar. Werden „class“- oder „delegateExpression“-Attribute verwendet, muss die entsprechende Bean das jeweilige Interface TaskListener oder ExecutionListener aus dem Paket org.activiti.engine.delegate mit einer entsprechenden „notify“-Methode (siehe Tabelle 3) implementieren.

Tabelle 3 Activiti Listener Interfaces

TaskListener	`void notify(DelegateTask delegateTask);`
ExecutionListener	`void notify(DelegateExecution execution) throws Exception`

Beispiel:

```
<sequenceFlow id="flow3" name="" sourceRef="servicetask1"
targetRef="endevent1">
      <extensionElements>
        <activiti:executionListener event="take"
       delegateExpression="${processOneEndListener}">
</activiti:executionListener>
      </extensionElements>
</sequenceFlow>
```

2.2.9 Funktionen der Activiti Engine

Die Activiti Engine arbeitet auf einer Datenbank mit vorgegebenen Tabellen und bietet Funktionalität in Form von sechs APIs an. Diese APIs sollen vorgestellt und wichtige Methoden aufgezählt werden.

2.2.10 Repository Service

Wie bereits erwähnt werden an die Activiti Engine Prozess Definitionen übergeben, was auch als Deployment bezeichnet wird. Dabei können in Form eines *.BAR Archivs mehrere Prozessdefinitionen oder einzelne XML-Dateien deployt werden. Dabei ist die Prozess-ID im XML entscheidend zur Identifikation. Wird ein Prozess mehrfach deployt, so wird bei erneutem Deployment die Definition als neue Version identifiziert und mit einer entsprechenden Versionsnummer versehen. Generell können alle Versionen eines Prozesses verwendet werden, solange eine Version nicht explizit deaktiviert wird. Die Definitionen lassen sich ihrem Deployment zuordnen, welches neben den Definitionen ebenfalls in der Datenbank mit Nummer und Timestamp gespeichert wird. Entsprechende Entitätsobjekte, welche Einträge in den Tabellen repräsentieren, lassen sich mit parametrisierbaren Queries mit Hilfe von Java-Methoden aus der Datenbank laden. Für ein Deployment muss mindestens eine korrekte Prozessdefinition vorliegen. Fehlerhafte Definitionen werden erkannt und nicht deployt.

2.2.11 RuntimeService

Der RuntimeService bietet Dienste für die Ausführung von Prozessen, wie das Starten neuer Prozessinstanzen, das Setzen und Lesen von Prozessvariablen, die manuelle Übertragung von Signalen und BPMN Events an Prozesse und das Löschen laufender Instanzen. Ein Prozess kann mit einer Liste von Variablen gestartet werden, die als Prozessvariablen gehalten werden. Man kann Prozesse der ID ihrer Prozessdefinition oder Prozess-ID aus der XML-Definition mit oder ohne Angabe der Version starten. Ohne Angabe der Versionsnummer wird die aktuellste aktive Version ausgewählt. Ein laufender Prozess wird in der Activiti Datenbank einer Tabelle gehalten und ist als „ProcessInstance“-Entität als

Java Objekt verfügbar. Diese Entität enthält Informationen über den Prozess, u.a. Start- und Endzeit, Initiator des Prozesses, Verweis auf die Prozessdefinition oder Gesamtdauer des Prozesses. Einige dieser Informationen sind vor Abschluss des Prozesses logischerweise nicht gesetzt. Der Prozesszustand wird durch Execution-Objekte repräsentiert. Jede Aktivität, sei es Gateway, Task oder Event, wird ein solches Objekt zugeordnet. Persistiert werden diese Objekte in der Datenbank nur, wenn die Aktivität nicht automatisch ausführbar ist, wie ein User Task, oder eine Gabelung des Sequence Flows, durch einen Gateway, vorliegt. Zu einem Zeitpunkt n im Prozessverlauf stellt die Menge seiner Executions seinen Zustand durch seine Aktivitäten dar. Nach Abschluss einer Aktivität fällt diese Instanz und ihr eventueller Datenbankeintrag weg. Der RuntimeService ermöglicht es, über Angabe der Execution, Prozessvariablen zu lesen und zu schreiben. Oben genannte Übertragung von Signalen und BPMN Events erfolgt ebenfalls unter Angabe der Execution und hat nur dann Auswirkungen, wenn die Aktivität auf ein solches Signal wartet. Weitere Funktionen wie das Löschen oder die vorrübergehende Deaktivierung laufender Prozesse erfolgt unter Angabe der ProcessInstance. Spezifische, parametrisierbare Queries zum Laden von Executions und ProcessInstances stehen ebenso zur Verfügung.

Prozessvariablen

Prozessvariablen in Activiti können sowohl Standard Java-Typen wie String oder Integer als auch eigene Datentypen sein. Eigene Datentypen müssen serialisierbar und dazu mit dem entsprechenden Interface java.io.Serializable gekennzeichnet sein. Der Schlüssel oder Bezeichner einer Variablen ist stets ein String. Variablen werden von Activiti in einer eigenen Datenbank gespeichert.

2.2.12 TaskService

Wie bereits erwähnt erfolgt eine manuelle Ausführung nur bei User Tasks. Der TaskService bietet für diese Tasks Bearbeitungsmöglichkeiten und Queries zum Laden von Taskinstanzen und Taskdefinitionen aus der Datenbank. In diesem Abschnitt ist mit Task daher immer der User Task gemeint. Die Queries sind wie in den anderen Services spezifisch für den Entitätstyp, in diesem Fall den Task-Typ. Einer Taskinstanz in Activiti werden dabei neben einer eindeutigen Nummer seine Execution, Prozessinstanz und der Name des Tasks im Prozessmodell zugeordnet.
Ein Task repräsentiert eine Aufgabe im Geschäftsprozess und wird von einer Person bearbeitet, in Activiti wird dieser als Assignee bezeichnet. Zu diesem Zweck existieren verschiedene Zuordnungsmöglichkeiten: direkte Zuordnung, Angabe von Kandidatenbenutzergruppen oder die Angabe einer Menge von Nutzern als Kandidaten. Entsprechende Kandidaten können dann als Grundlage für die letztendliche Zuordnung verwendet werden. Der Task kann weiterhin einen Eigentümer haben, Owner genannt, und dem Task muss immer eine Priorität zugeteilt werden. All diese Zuordnungen können auf XML-Ebene, mit dem TaskService oder in Kombination beider erfolgen. Der Task kann weiterhin mit dem TaskService abgeschlossen, an einen User delegiert oder gelöscht werden. Es existiert auch eine Methode um einen Task nach Bearbeitung aber vor Abschluss an den Eigentümer zur Kontrolle zu übertragen.

Sollte ein Task Form Properties besitzen, wie in Abschnitt 2.2.7 beschrieben, kann ein Task zusammen mit entsprechenden Daten auch über den FormService (siehe nächster Abschnitt) abgeschlossen werden. Dies ist jedoch nicht zwangsweise erforderlich.

2.2.13 FormService

Dieser Service bietet Dienste, um Form Properties (siehe Abschnitt 2.2.7) für den Prozessstart oder einen User Task zur Verfügung zu stellen. Diese Form Properties werden in Listenform zur Verfügung gestellt. Jede Property besitzt dabei entsprechend definierte Attribute und kann in der Anwendung für die Erstellung von grafischen Schnittstellen verwendet werden.
Es ist möglich Prozesse mit dem FormService zu starten und Tasks abzuschließen. Diese Möglichkeit wird zur Verfügung gestellt, um Werte für Form Properties direkt aus Eingabe Masken in einer Liste von Schlüssel-Wert-Paaren an die Engine zu übergeben und für den Start bzw. Abschluss zu verwenden. Diese Listen werden von der Engine dabei als Prozessvariablen gespeichert.

Tabelle 4 Engine Interaktionen: Prozess starten & Task abschließen

Prozessstart	`ProcessInstance submitStartFormData(String processDefinitionId, Map<String, String> properties);`
User Task Abschluss	`void submitTaskFormData(String taskId, Map<String, String> properties);`

2.2.14 HistoryService

Aus rechtlichen oder statistischen Gründen kann es erforderlich sein Daten über Prozesse über deren Lebensdauer hinweg zu speichern. Activiti kennt dabei verschiedene Ebenen der Historisierung, um entsprechend keine oder verschieden detailierte Daten zu speichern. Dies kann über die Konfiguration der Engine bestimmt werden. Bei voller Historisierung werden Daten über Prozessinstanzen, alle Aktivitäten, Tasks und Details gespeichert, welche über eigene Queries aus der Datenbank geladen werden können. Da Tasks Aktivitäten sind, werden sie in diesem Fall doppelt historisiert. Als Details werden dabei betrachtet:

- Das Setzen oder Verändern von Prozessvariablen
- Übergänge (Sequenceflow)
- Zuweisungen von User Tasks an User
- Übergebene Form Properties

In der Regel werden dabei neben den jeweiligen Inhalten Daten über Start- und Endzeitpunk gespeichert.

2.2.15 IdentityService

Activiti bietet mit dem IdentityService ein einfaches System für User und Gruppen an. Beide können über den Service angelegt und verwaltet werden. Es lassen sich dabei User Gruppen

zuordnen, wobei ein User zu mehreren Gruppen gehören kann. Dies ist über eine Membership-Entität realisiert. Ein User hat Attribute wie eine eindeutige Usernummer, Vor- und Nachnamen, E-Mail und ein Password. Einer Gruppe kann ein Name und ein Gruppentyp zugeordnet werden. Die User und Gruppen können als Kandidaten oder Bearbeiter für User Tasks fungieren wie in Abschnitt 2.2.12 beschrieben.

Verwendung des IdentityService in Verbindung mit dem TaskService

Der TaskService bietet die Möglichkeit Tasks für bestimmte User oder Gruppen per Query zu laden. Es können zum Beispiel für einen angemeldeten Nutzer Tasks geladen werden, für die er ein Kandidat ist durch Zugehörigkeit zu einer Gruppe. Dieser Task kann dann für den User beansprucht werden.

Beispiels
Alle Tasks für die Gruppe "sales":

```
taskService.createTaskQuery().taskCandidateGroup("sales").singleResult();
```

Task einem Benutzer zuordnen:

```
taskService.claim(task.getId(), user.getId());
```

2.2.16 Konfiguration der Activiti Engine

Für die Verwendung der Activiti Engine muss eine Implementierung gewählt werden. Vor der Verwendung muss die Engine konfiguriert werden. Dabei können viele Attribute konfiguriert werden. Die Konfiguration erfolgt mit Hilfe einer ProcessEngineConfiguration. Die Standardklasse dafür ist org.activiti.engine.ProcessEngineConfiguration. Die Attribute können über Setter oder über eine Konfigurationsdatei in Form einer Spring-Bean-Definition bestimmt werden. Im Folgenden sollen beide Varianten mit den wichtigsten Attributen, wie einer Datenbankverbindung, an Beispielen gezeigt werden.

Konfiguration mit Datei:

```
ProcessEngine processEngine = ProcessEngineConfiguration
                            .createProcessEngineConfigurationFromResource(

      "/activiti.cfg.xml").buildProcessEngine();
```

```
<!-- process engine configuration -->
<bean id="processEngineConfiguration"
class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">
            <property name="dataSourceJndiName" value="jdbc/Database" />
            <property name="history" value="full" />
      </bean>
</beans>
```

Konfiguration ohne Datei:

```
ProcessEngineConfiguration engConf = ProcessEngineConfiguration

      .createStandaloneProcessEngineConfiguration();
engConf.setDataSourceJndiName(„jdbc/Database“);
engConf.setHistory(„full“);
engine = cdiEngineConf.buildProcessEngine();
```

In den Beispielen wird eine JNDI-Datenquelle verwendet, wie sie in einem Application Server erstellt werden kann. Es ist auch möglich eine eigene Verbindung aufzubauen, indem man entsprechend eine JDBC Verbindung konfiguriert, mit URL der Datenbank, Benutzername, Password und Angabe der Treiberklasse.[7]
Die Methode *createProcessEngineConfigurationFromResource* verwendet dabei die ProcessEngineConfiguration-Implementierung aus der Konfigurationsdatei. Das Beispiel ohne Datei verwendet die Standard-Implementierung. Würde der Java-Code aus dem Beispiel mit Datei mit den Änderungen aus Abschnitt 5 verwendet werden, würde entsprechend die *CdiJtaProcessEngineConfiguration* Implementierung verwendet werden.

2.2.17 Implementierungen der Activiti Engine

Die Standard Activiti Engine-Implementierung bietet alle im Abschnitt 3 genannte Funktionalität. Diese unterscheidet sich von Engine zu Engine nicht. Allerdings wurde in Abschnitt 2.1 die Verwendung von Spring- und CDI Named Beans aufgezeigt. Diese ist mit der Standard Implementierung nicht möglich. Es existieren gesonderte Module die eine solche Implementierung für Spring (org.activiti.spring) und für CDI (org.activiti.cdi von Camunda) zur Verfügung stellen.

Die einfache Konfiguration unterscheidet sich dabei nicht von der Standard Engine, es sei denn man möchte z.B. die Transaktionsverwaltung von Spring oder einer anderen JTA-Implementierung nutzen. Dann muss dieser in der Konfigurationsdatei aus dem Beispiel im vorigen Abschnitt als Bean definiert sein und als Property der processEngineConfiguration-Bean gesetzt werden.

Beispiel für die Verwendung des Glassfish-Transaktionsmanager:

```
<bean id="transactionManager"
      class="org.springframework.jndi.JndiObjectFactoryBean">
   <property name="jndiName"
value="java:appserver/TransactionManager"></property>
   <property name="resourceRef" value="true" />
</bean>
…
<bean id="processEngineConfiguration"
             class="org.activiti.cdi.CdiJtaProcessEngineConfiguration">
       <property name="transactionManager" ref="transactionManager" />
…
```

Mit dieser Konfiguration können in expression- und delegateExpression-Attributen an allen Stellen, an denen diese verwendet werden können, entsprechend CDI Named Beans verwendet werden. Die Transaktionsverwaltung wird vom Application Server Glassfish übernommen.

[7] Weitere Einstellungsmöglichkeiten können in „Activiti in Action“ (siehe [10]) und auf der Activiti-Website (siehe [13]) gefunden werden

2.3 Private Krankenversicherung [8]

In diesem Abschnitt soll eine Einführung in das Thema Private Krankenversicherungen gegeben werden. Diese dient dem Verständnis des fachlichen Hintergrunds und der Leistungsabrechnung selbst, die als Fallbeispiel für einen realen Prozess zur Ausführung in der Activiti Process Engine dient.

2.3.1 Aufgaben der Privaten Krankenversicherung

In Deutschland besteht eine grundsätzliche Krankenversicherungspflicht, welche eine gesetzliche Krankenversicherung (GKV) bei einer Krankenkasse vorsieht, die einen Zweig der Sozialversicherung darstellt. Diese Versicherungspflicht entfällt jedoch für bestimmte Personen, beispielsweise für Freiberufler, Beamte oder Arbeitnehmer mit einem Einkommen über einem bestimmten Grenzwert. Personen, die von der Versicherungspflicht befreit sind, können sich entscheiden, ob sie sich freiwillig gesetzlich versichern wollen oder Mitglieder einer privaten Krankenversicherungen (PKV) werden, wobei der Versicherungsschutz dabei von gewählten Tarifen abhängt. In Deutschland gibt es (Stand 2009) 49 Anbieter solcher Versicherungen, die unter dem PKV-Verband organisiert sind. Dabei bietet die PKV nicht nur Krankenvollversicherung an, sondern auch Zusatzversicherungen für gesetzliche Versicherte, sowie Teilversicherungen für Beamte (Beihilfeversicherung).
Im Gegensatz zum Standardfall in der GKV (Ausnahme Wechsel von PKV zu GKV) ist die PKV nicht verpflichtet einen Antrag auf Versicherungsschutz anzunehmen. Die Beitragsberechnung für die Versicherten geschieht in der GKV auf Basis des Bruttoeinkommens, während in der PKV der Anfangsbeitrag von Gesundheitszustand, Alter, Geschlecht und Leistungsumfang abhängt, welcher sich im Laufe der Versicherung verändern kann. Eine Beitragsanpassung erfolgt dabei kollektiv auf Basis von Gesamtstatistiken und nicht in Folge individueller Kosten. Die Beitragsberechnung unterliegt nach dem Versicherungsaufsichtsgesetz gesetzlicher Aufsicht.
Änderungen im Versicherungsvertragsgesetz haben 2009 zu einer Pflichteinführung eines Basistarifs (§ 315 Abs. 4 Sozialgesetzbuch) für alle Privaten Krankenversicherer geführt. Dieser Tarif schreibt einen ähnlichen Versicherungsschutz wie die GKV vor. Dabei dürfen Anträge von grundsätzlich berechtigten Personen nicht abgelehnt werden, es sei denn der Person wurde der Versicherungsschutz zuvor von einem Versicherer auf Grund von Betrug oder arglistiger Täuschung gekündigt.

2.3.2 Versicherungsschutz

In der PKV gibt es folgende Versicherungsarten: Krankheitskostenversicherung, Krankentagegeldversicherung und Pflegeversicherung. Die Krankheitskostenversicherung bietet Schutz in Form einer Schadensversicherung vor Kosten, die direkt durch Erkrankungen und Unfälle entstehen, wie z.B. ärztliche Beratung oder Behandlung oder Krankenhausaufenthalte (jedoch nicht Verdienstausfälle, die damit in Verbindung stehen). Liegt ein vom Umfang her mit GKV vergleichbarer Versicherungsschutz vor, so spricht man

[8] Quellen für diesen Abschnitt sind „Spezielle Versicherungslehre Band 2“ von H. Eichenauer et al. (siehe [12]) und „Ausbildungsliteratur. Private Kranken- und Pflegeversicherung“ von P. Bertram und P. Schlinck (siehe [11])

von einer substituiven Krankenversicherung. Der Versicherungsschutz ist, mit Ausnahme des Basistarifs, von der Tarifwahl abhängig. Tarife können dabei grundsätzlich folgender Art sein: [9]

- 100%-Tarife
  - Bieten vollen Krankenversicherungsschutz, je nach Wahl des Versicherten mit Selbstbeteiligung für die Kostenübernahme
- Quotentarife
  - Prozentuale Kostenübernahme auf Basis eines vom Versicherten gewählten Wertes
- Zusatz- und Ergänzungstarife
  - Bieten zusätzliche Leistungen wie Chefarztbehandlung oder ergänzende Leistungen wie die Übernahme von Kosten für Sehhilfen an

Der Basistarif muss in Form eines 100%-Tarifs sowie in einer Beihilfe-Variante verfügbar sein. Tarife mit Selbstbeteiligungen können dabei verschiedener Form sein. Sie legen pro Abrechnungsjahr fest, dass für bestimmte Leistungen entweder bis zu einem festgelegten Betrag der Versicherte selbst aufkommen muss (Abzugsfranchise) oder ab einem festgelegten Betrag keine (Maximalsystem) oder eine prozentuale (Maximal-Prozentual) Erstattung stattfindet.
Tarife decken dabei eine oder mehrere der folgenden Leistungsarten ganz oder teilweise ab:

Tabelle 5 Leistungsarten

Ambulante Behandlung	Stationäre Behandlung	Zahnärztliche Behandlung
- Ärztliche Beratung - (Vorsorge-) Untersuchungen - Arzneimittel - Transporte - Entbindungen - Heilpraktiker	- Ärztliche Leistungen - Unterbringung und Verpflegung - Krankenhaustransporte - Wahlleistungen (z.B. Einzelzimmer)	- Zahnärztliche Leistungen - Zahnersatz - Kieferorthopädie

Aufzählung der Leistungsarten und Teilaufstellung von Details [10]

Dabei gibt es für Ärzte (GOÄ), Zahnärzte (GOZ) und Heilpraktiker (GOH) Gebührenordnungen, die festlegen welche Leistungen zu welchem Betrag abgerechnet werden. Leistungserbringer können erbrachte Leistungen jedoch mit einem Faktor versehen, um Spielraum für besondere Aufwände zu gewähren. Dabei gibt es je nachdem, ob eine persönliche ärztliche, medizinisch-technische oder Laborleistung vorliegt, Regelspannen und Höchstwerte für die Wahl des Faktors, wobei außerhalb der Regelspanne eine gesonderte Begründung vorgelegt werden muss. Tarife können dabei eigene Höchstwerte für Faktoren festlegen. Mit DRG (Diagnoses Related Groups) Fallpauschalen liegt ein ähnliches System für die Abrechnung stationärer Leistungen vor, das ebenfalls ein Gewichtungssystem vorsieht.

[9] Siehe [11, S. 56-57]
[10] Entnommen aus [11, S. 52-55]

Beispiel: Die Regelspanne für persönliche ärztliche Leistungen liegt zwischen dem 1-fachen und 2,3-fachen Satz der jeweils vorgesehenen Gebühr. Darüber hinaus kann der Leistungserbringer bis zum 3,5-fachen Satz abrechnen, jedoch erfordert dies eine Begründung die angefochten werden kann.

Die Krankentagegeldversicherung und die Pflegeversicherung sind im Gegensatz zur Krankheitskostenversicherung Summenversicherungen. Sie decken einen abstrakten Bedarf (z.B. als Tagessatz) ab, der in der Versicherung festgelegt ist und sich damit vom konkreten Fall unterscheiden kann. Dabei tritt eine Krankentagegeldversicherung bei vorrübergehender Arbeitsunfähigkeit in Kraft und ersetzt Verdienstausfälle. Eine Pflegeversicherung deckt die Betreuungskosten ab, wenn durch Krankheit oder Unfall Pflegebedürftigkeit entstanden ist. Pflegebedürftigkeit liegt dann vor, wenn die Verrichtung alltäglicher Tätigkeiten auf Dauer (mindestens 6 Monate) durch körperliche oder geistige Beeinträchtigung nicht möglich ist.

Zusätzlich zu den Tarifbedingungen und Beiträgen können für bestimmte Leistungen im tariflichen Rahmen besondere Versicherungsbedingungen vom Versicherer zum Versicherungsbeginn festgelegt werden. Diese sind entweder als Risikozuschlag oder als Leistungsausschluss definiert und werden für Leistungen vorgenommen, bei denen durch Vorerkrankung oder allgemeinen Gesundheitszustand für den Versicherer ein zusätzliches bis unkalkulierbares Risiko besteht. Ohne die Zustimmung zu diesen Bedingungen durch den potentiellen Versicherungsnehmer kommt kein Versicherungsvertrag zustande. Diese Bedingungen können im Laufe der Versicherung zurückgenommen werden. Solche besonderen Versicherungsbedingungen dürfen nicht bei einer Versicherung im Rahmen des Basistarifs vorgenommen werden.

2.3.3 Versicherungsvertrag

Eine Versicherung wird mit Abschluss eines Vertrages vereinbart, welcher auch Police genannt wird. Der Vertrag besteht zwischen dem Versicherungsnehmer und dem Versicherer, wobei zu einem Vertrag mehrere Versicherte Personen mit verschiedenem Versicherungsschutz gehören können und der Versicherungsnehmer nicht unbedingt eine Versicherte Person ist. Gegenstand des Vertrages ist der vereinbarte Versicherungsschutz gegen Prämie, die dabei geltenden AVBs (Allgemeine Versicherungsbedingungen, Vorlage vom PKV Verband), die besonderen Versicherungsbedingungen und Obliegenheiten. Obliegenheiten sind im VVG (Versicherungsvertrags Gesetz) definiert und legen z.B. eine Frist für die Meldepflicht von Erkrankungen oder Arbeitsunfähigkeit vor. Des Weiteren muss der Versicherer seine Informationspflicht nach VVG erfüllen (allgemeine und zusätzliche Versicherungsinformationen, Produktinformationsblätter). Versicherte müssen vorvertraglich alle verlangten Angaben, u.a. zum Gesundheitszustand (vorvertragliche Anzeigepflicht), im Antrag unter Versicherung besten Gewissens machen.
Der Beginn des Versicherungsschutzes (materieller Beginn) ist grundsätzlich das im Vertrag vereinbarte Datum (technischer Beginn), es sei denn der Abschluss des Vertrages geschieht nach diesem Termin (formeller Beginn). Dabei gilt, dass Versicherungsfälle, welche nach dem technischen aber vor dem formellen Beginn liegen, auch wenn sie bis nach diesen reichen, grundsätzlich nicht versichert sind.
Obwohl weitestgehend auf Wartezeiten von Seiten des Versicherers verzichtet wird, können solche in Form einer Karenzzeit vereinbart werden. Versicherungsfälle, die in dieser

leistungsfreien Zeit auftreten und bis nach dieser reichen, sind für den nachreichenden Teil versichert. Zweck einer solchen Wartezeit ist meist Risikominimierung für Krankheitsbilder die sich vor Beginn schon abzeichneten. Wartezeiten dauern im allgemeinen Fall 3 Monate oder im speziellen Fall 8 Monate ab technischem Beginn. Weitere leistungsfreie Zeiten können sich durch qualifizierten Zahlungsverzug der Prämien nach der zweiten Mahnung ergeben. Auftretende Versicherungsfälle in diesen Zeiträumen sind in diesem Fall auch nicht teilversichert für Leistungen, die bis nach Zahlung ausstehender Beträge laufen. Ein Versicherungsfall beginnt dabei mit dem ersten Behandlungstag und endet nach Feststellung, dass keine Behandlungsbedürftigkeit oder Arbeitsunfähigkeit (Krankentagegeldversicherung) mehr vorliegt. Versicherungsverträge im Rahmen der PKV sind in der Regel unbefristet.

2.3.4 Leistungsabrechnung

Erbrachte Leistungen werden vom Leistungserbringer generell dem Versicherten in Rechnung gestellt, welcher wiederrum diese Kosten bei bestehendem Leistungsanspruch vom Versicherer erstattet bekommt. Eine Ausnahme hierbei bilden stationäre Kosten, da diese in ihrer Höhe in der Regel nicht vom Versicherten bezahlt werden können und daher meist direkt mit dem Krankenhaus abgerechnet werden. In erstem Fall erhält der Versicherer die Rechnung über die erbrachten Leistungen mit Diagnose nach jeweiliger Gebührenordnung, DRG oder sonstiger Angaben und zahlt, nach Prüfung der Leistungspflicht, den jeweils tariflich festgelegten Betrag. Eine Leistungspflicht liegt dabei nicht vor, wenn:

- eine Verletzung vertraglich festgelegter Obliegenheiten, wie eine Nachuntersuchung bei einem vom Versicherer festgelegten Arzt, vorliegt
- die Leistung nicht tariflich abgedeckt ist
- Leistungsausschlüsse für die Leistung gelten
- Wartezeiten vorliegen
- der materielle Beginn der Versicherung nach dem Leistungsdatum liegt
- eine Verletzung vorvertraglicher Anzeigepflicht vorliegt
- ein qualifizierter Zahlungsverzug besteht
- Krankheit oder Unfall auf Vorsatz beruhen
- Krankheit, Unfall und Folgen durch Kriegsereignisse verursacht wurden
- Nicht alle vom Versicherer verlangten Nachweise erbracht worden sind

Liegt eine Leistungspflicht vor hängt die Erstattungshöhe von tariflich vereinbarten Erstattungssätzen oder Selbstbeteiligungen ab. Außerdem kann eine Vorleistung von einer weiteren PKV oder GKV vorliegen und sich auf die Erstattungshöhe auswirken. Der Versicherungsnehmer erhält im Anschluss ein Dokument, das Auskunft über die Leistungsabrechnung gibt und erhält gegebenenfalls eine Auszahlung. In besonderen Fällen kann eine einmalige Erstattung auf Grund von Kulanz erfolgen, sollte keine Leistungspflicht vorliegen.

2.4 Begriffserklärung Modellgetriebene Softwareentwicklung [11]

Modelle spielen seit jeher in der Softwareentwicklung eine Rolle. Seien es Flowcharts oder UML-Diagramme, in den meisten Fällen dienen sie nur als Dokumentation oder Grundlage in der Entwurfsphase. In letzteren Fällen kann man von modellbasierter Entwicklung sprechen. Da ein Modell meist präzise die Anforderungen darstellt und daher auch von Fachkräften in einer entsprechenden Form erstellt wird, besteht das Risiko, dass bei der Umsetzung des Modells in Software eine Diskrepanz zwischen dieser und dem Modell entsteht. Solche Diskrepanzen führen in der Regel zu Diskussionen zwischen Entwicklern und Benutzern und sorgen oft für kostspielige Nacharbeiten zur Erfüllung der Anforderungen. Dies kann entsprechend das Verhältnis beider Seiten stören oder gar zum Scheitern von Projekten führen. Die modellgetriebene Softwareentwicklung oder Model Driven Development (MDD) verfolgt daher den Ansatz, eine direkte Verbindung von Modell und Software zu schaffen, indem das Modell direkt in der Software oder als Input für einen Softwaregenerator verwendet wird. Ein Beispiel für modellgetriebene Softwareentwicklung ist die direkte Verwendung von BPMN 2.0 Modellen, mit ihrer standardisierten XML-Form zur Ausführung in einer Process Engine, welche BPMN implementiert.

2.5 Technische Frameworks

2.5.1 Spring[12]

Das Spring-Framework bietet eine breite Menge an Funktionalität entsprechend Konzepten aus der Softwareentwicklung wie Dependency Injection, MVC oder Aspekt orientierte Programmierung für Java an. Für diese Arbeit ist nur die Dependency Injection relevant und soll an dieser Stelle generell sowie die Umsetzung in Spring erläutert werden.
In der Softwareentwicklung bestehen oftmals Abhängigkeiten zu Modulen, deren Funktionalität benötigt wird, die fest im Code definiert sind. Diese Abhängigkeiten können dabei durch Schnittstellen entkoppelt werden. Ein Beispiel dafür ist die von Sun definierte Java Persistence API und ihre Interfaces, für die mehrere Implementierungen existieren. Wechselt man aber die verwendete Implementierung, muss dies direkt im Code an jeder entsprechenden Stelle geändert werden. Dies führt generell zu erhöhtem Wartungsaufwand und Fehlerpotential, sollte eine Stelle vergessen werden. Das Konzept der Dependency Injection bedeutet, dass solche Abhängigkeiten von Schnittstellen extern in einem Dienst definiert und Instanzen entsprechender Implementierung verwaltet werden. Benötigt ein Programm nun diese Schnittstelle, dann wird die Abhängigkeit in entsprechender Implementierung injiziert. Ändert sich die Implementierung muss dies nur einmalig im externen Dienst eingetragen werden. Damit muss nur entsprechende Kenntnis vorhanden sein wie das entsprechende Modul im externen Dienst verfügbar ist.

[11] Abgeleitet von Grundlagen in [8] und [9]
[12] Quelle für diesen Abschnitt ist „Spring 3. Framework für die Java-Entwicklung“ von E. Wolff (siehe [2])

Die Umsetzung dieses Konzepts in Spring für Java erfolgt mit einer Bean Definition in XML-Form, in der sogenannte Spring Beans definiert werden können. Diese können einen beliebigen Namen, aber auch einfach den Namen des Interfaces als Bezeichner, haben und verweisen auf den vollqualifizierten Paketpfad der Implementierung. Das Spring-Framework muss diese Definitionsdatei kennen und stellt entsprechende Dienste zur Verfügung, diese Beans bereitzustellen, und verwaltet die Instanzen.
Dies soll an einem einfachen Fall, im Folgenden Beispiel, mit einer Schnittstelle (Java Interface) und einer Implementierung (Java Klasse) gezeigt werden. Weitere Funktionen und Informationen können entsprechender Literatur (siehe z.B. [2]) entnommen werden.

Schnittstelle: **interface** org.dependencyInjection.CalculationService
Implementierung: **class** org.dependencyInjection.PreciseCalculationService

spring-beans.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:tx="http://www.springframework.org/schema/tx"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
                            http://www.springframework.org/schema/beans/spring-beans.xsd
                            http://www.springframework.org/schema/context
                            http://www.springframework.org/schema/context/spring-context-
3.0.xsd
                            http://www.springframework.org/schema/tx
                            http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">

        <bean id="calcService" class="org.dependencyInjection.PreciseCalculationService"/>

</beans>
```

Dependency Injection:

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("/spring-beans.xml");
CalculationService calculationService = applicationContext.getBean(„calcService“);

2.5.2 Maven [13]

Maven ist ein Java Framework, das Dienste anbietet, um Projektabhängigkeiten zu verwalten, Projekte zu kompilieren und Auslieferungsartefakte zu erstellen. Die Konfiguration dafür, auf Projekt- und Projektmodulebene, wird dabei durch Project Object Models (POM, XML-Dateien) vorgenommen. Diese POM-Dateien können generiert werden und über eine Oberfläche bearbeitet werden, z.B. in Eclipse, weshalb ihre genaue Struktur an dieser Stelle nicht aufgezeigt wird. In dieser Arbeit ist primär die Verwaltung von Projektabhängigkeiten relevant, was an einem Fallbeispiel gezeigt werden soll. Die Verwaltung von Projektabhängigkeiten zu anderen Projekten oder Frameworks erfolgt anhand von dependency-Einträgen. Diese verweisen auf das entsprechende Artefakt, welches benötigt wird, gegebenenfalls mit Vorgabe einer bestimmten Version. Artefakte werden in Maven Repositories gehalten und werden von dort, bei Ausführung eines Maven

[13] Quelle für diesen Abschnitt ist „Maven 2. Eine Einführung, aktuell zur Version 2.0.9“ von K. U. Bachmann (siehe [14])

Goals, heruntergeladen. Die Verwaltung solcher Repositories obliegt dem jeweiligen Besitzer, wobei es ein zentrales Repository von Maven selber gibt, das im Standardfall verwendet wird. Angesprochene Maven Goals dienen unterschiedlichen Zwecken, sind aber im Ursprung zum Kompilieren, Erzeugen von Artefakten und Deployment dieser Artefakte vorgesehen. Das Goal „package“ führt zum Beispiel zur Erzeugung des jeweiligen Java Artefakts, wie einer JAR-, WAR- oder EAR-Datei. Goals werden im Normalfall über die Kommandozeile ausgeführt oder über ein Plugin für entsprechende Entwicklungstools.

Beispiel:

In einem Projekt wird Spring in der Version 3.1.0.RELEASE verwendet. Die Abhängigkeit wird über Maven verwaltet.

POM.xml

```
…
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-beans</artifactId>
	<version>3.1.0.RELEASE</version>
</dependency>
…
```

Ausführung des Goals

Die Ausführung des Maven Goals erfolgt in der Kommandozeile in dem Verzeichnisknoten, in dem die POM-Datei liegt:

mvn package

3 Fallbeispiel Leistungsabrechnung

Als Fallbeispiel soll ein fiktiver Leistungsabrechnungsprozess erstellt werden, der im Anschluss, im Zusammenspiel mit der Activiti Process Engine, als IT-gestützter Geschäftsprozess in einer Applikation ausführbar gemacht werden soll. Aus Gründen fachlicher Komplexität wird hierbei vereinfacht. Die Leistungsabrechnung eignet sich gut für die Ausführung durch eine Process Engine, weil:

- es ein täglich und in großen Mengen auftretender Prozess ist, der um Kosten gering zu halten standardisiert und schnell ausgeführt werden sollte.
- es ein Prozess ist, der unter Umständen zur Auszahlung großer Summen führen kann und somit eine korrekte Prüfung der Zahlungsvoraussetzung und Berechnung des Auszahlungsbetrags nach den gesetzlichen und vertraglichen Regeln erforderlich ist.
- bei bestimmten Voraussetzungen im jeweiligen Fall eine (automatische) Delegierung an einen medizinischen Rechnungsprüfer oder Vorgesetzten erfolgen muss.

Das vorgestellte fachliche Modell unterscheidet sich von dem im späteren Verlauf verwendetem Prozessmodell für die Activiti Engine, da in dieser Elemente wie Pools und Message Flows sowie die hierarchische Darstellung von Subprozessen nicht unterstützt werden. Das Modell muss entsprechend für die technische Verwendung verändert werden. Es werden Elemente entfernt werden und Subprozesse durch Call Activities ersetzt. Die Verwendung von Call Activities erhält die Übersichtlichkeit und erfüllt denselben Zweck, entspricht aber nicht der korrekten semantischen Verwendung.
Im weiteren Verlauf des Kapitels wird das Prozessmodell vorgestellt. Dabei wird nicht auf die Erstellung des Modells eingegangen. Die grundlegende Methode wurde im Grundlagenkapitel im Abschnitt BPMN vorgestellt. Elemente, die nicht von Activiti unterstützt werden, werden im letzten Abschnitt des Kapitels behandelt.

3.1 Top-Level-Prozess

Aus Gründen der Anschaulichkeit wird der Prozess hierarchisch modelliert. Größere, zusammengehörige Prozessabschnitte werden daher als Subprozesse modelliert und explizit erläutert. Der Top Level Prozess stellt dabei den generellen Ablauf dar.

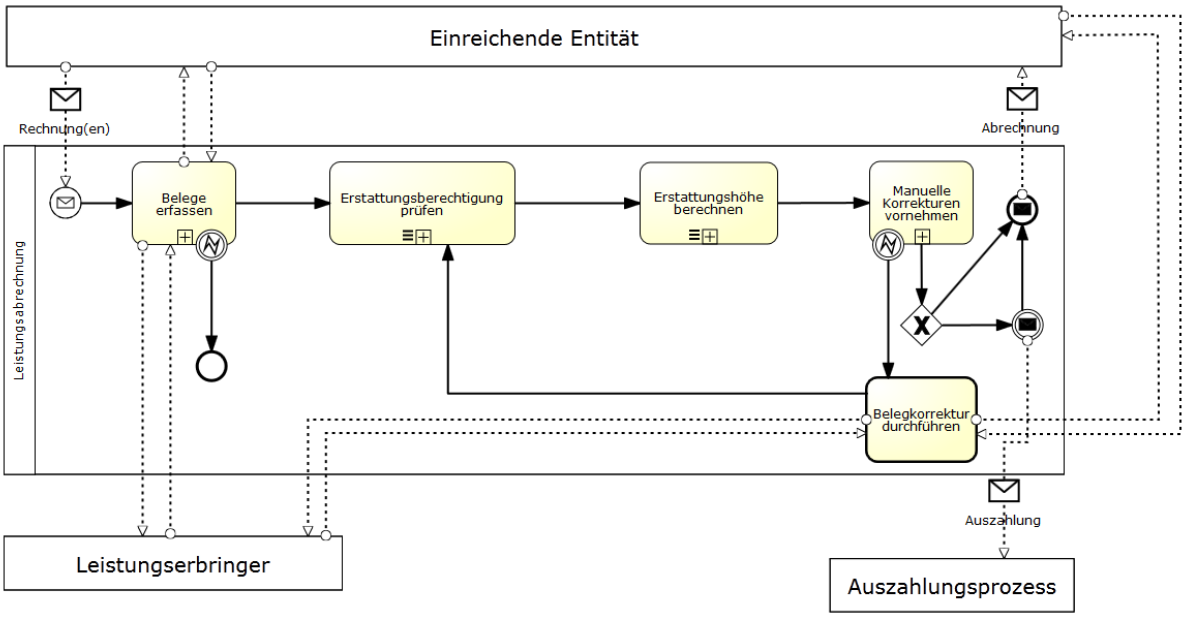


Abbildung 2 Top-Level-Prozessmodell Leistungsabrechnung

Die Leistungsabrechnung läuft in vier Schritten ab, die im Modell in den Subprozessen „Belege erfassen“, „Erstattungsberechtigung prüfen“, „Erstattungshöhe berechnen“ und „Manueller Eingriff vornehmen“ zu sehen sind. Eine Leistungsabrechnung ist ein Prozess, der ausgeführt wird, wenn der Versicherungsnehmer, eine Versicherte Person oder ein Krankenhaus eine Rechnung einreicht. Der Versicherer führt daraufhin eine Abrechnung durch, die zeitnah zur Einreichung erfolgt. Der genaue Zeitpunkt wird dabei vom zuständigen Bearbeiter gewählt. Der Auslöser ist also das Einreichen der Rechnungen und entsprechend wird dies mit einem Message Start Event modelliert. Der Message Flow geht dabei von einer externen Entität aus, welche schlicht und ohne weitere Differenzierung als „Einreichende Entität“ bezeichnet wird.
Es ist möglich, dass mehrere Rechnungen auf einmal oder im selben Zeitraum eingereicht werden, daher kann eine Abrechnung mehrere Belege abrechnen. Diese werden im gleichnamigen Subprozess erfasst, wobei bei Feststellung einer Anzeigepflichtsverletzung der Prozess vorzeitig beendet wird. Das gilt auch, wenn mehrere Belege abgerechnet werden, da eine solche Verletzung zu Vertragsänderungen oder gar Kündigung führen kann. Dieses Verhalten wird durch ein Error Boundary Event dargestellt, welches zu einem End Event führt. Im Verlauf der Erfassung kann es ebenfalls erforderlich sein weitere Auskünfte über die Behandlung vom Behandler oder Behandelten einzufordern. Dies wird mit Message Flows zwischen Prozess und entsprechenden Entitäten modelliert.
Im Normalfall folgt auf die Erfassung die Prüfung der Erstattungsberechtigung. Diese Prüfung muss sequentiell pro Beleg durchgeführt werden. Die Prüfung führt zu einer Zuordnung von leistenden Tarifen zu den jeweiligen Rechnungsposten. Die Berechnung der Erstattungshöhe basierend auf diesen Tarifen erfolgt im nächsten Subprozess. Im Anschluss besteht die Möglichkeit manuelle Änderungen vorzunehmen um die Erstattungsbeträge auf Basis eventueller Kulanzen oder anderweitiger Gründe zu korrigieren. Wird in diesem Prozessschritt festgestellt, dass eine fehlerhafte Erfassung der Belege vorliegt, wird ein Error Boundary Event ausgelöst. Dieser Event führt dann zu einer entsprechenden Call Activity zur Korrektur der Belege, wobei erneut weitere Auskünfte notwendig sein können. Die Korrektur muss als Call Activity modelliert werden, da sie an anderer Stelle in einem Subprozess ebenfalls Verwendung findet. Tritt der Korrekturfall ein, muss erneut eine Prüfung und Berechnung der Erstattung stattfinden. Bei normalem Abschluss der Tätigkeit, mit oder ohne tatsächliche Korrektur, erfolgt ein Auszahlungs-Auftrag an einen gesonderten Prozess in Höhe des Erstattungsbetrags. Dieser Auftrag wird durch ein Message Intermediate Event dargestellt, welcher eine „Auszahlung“ an den Prozess überträgt. Dies ist nur erforderlich, wenn der Erstattungsbetrag nicht „0“ ist, ansonsten wird direkt zum Ende übergegangen und ein Abrechnungsdokument an die Einreichende Entität übermittelt.

3.2 Subprozesse

Belege erfassen

Eingereichte Rechnungen werden mit Zeitraum, Leistungserbringer, Art der Behandlung (stationär, Zahnarzt etc.), Gesamtsumme und Versicherter Person erfasst. Die Rechnung besteht aus mindestens einer Rechnungsposition. Rechnungspositionen stellen einzelne Leistungen mit jeweiligen Kosten dar. Diese entsprechen Gebührenordnungsziffern, DRG-Nummern oder sonstigen nicht katalogisierten Leistungen. Die Positionen werden entsprechend mit Kennung, Zeitraum und Rechnungsbetrag erfasst. Nach der Erfassung einer Rechnung mit seinen Positionen erfolgt eine manuelle Prüfung der Eingaben. Dabei kann ein Benutzer prüfen, ob er Fehler bei der Eingabe gemacht hat oder eventuelle Auskunftsbedarf besteht. Liegt ein Korrekturbedarf vor, so wird erneut die aus die aus dem Top-Level-Prozess bekannte Call Activity dafür verwendet. Dabei besteht immer die Möglichkeit alle Belege und Positionen zu korrigieren. Im Fall eines Auskunftsbedarfs wird ein User Task verwendet, um entsprechende Kommunikation vorzunehmen und Auskünfte einzuholen. Nach Abschluss beider Tätigkeiten folgt eine erneute Prüfung der Belege. Diese

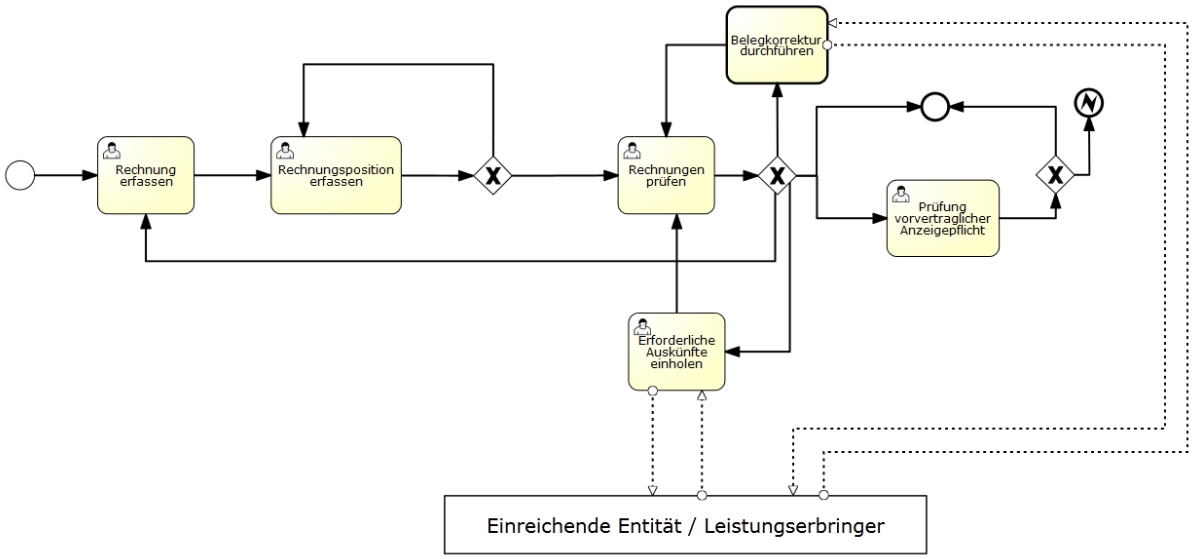


Abbildung 3 Sub-Prozess-Modell Belegerfassung

kann wiederrum zu erneuten Korrekturen oder Auskunftsanfragen führen. Ist alles korrekt erfasst und sind alle Informationen vorhanden kann aus der Prüfung das Ende des Subprozesses folgen. Besteht jedoch der Verdacht auf eine Verletzung vorvertraglicher Anzeigepflicht muss dieser Verdacht geprüft werden. Wird eine Verletzung festgestellt folgt ein fehlerhafter Abschluss. Dieser wird im Top-Level-Prozess behandelt.

Belegkorrektur durchführen

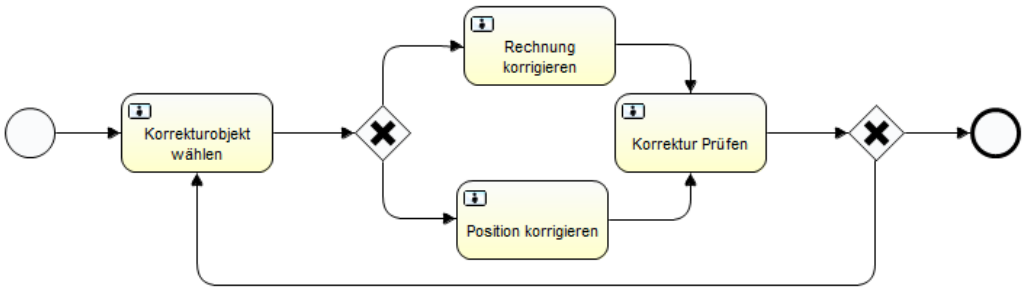


Abbildung 4 Sub-Prozess-Modell Belegkorrektur

Ist eine Belegkorrektur erforderlich wird diese auf Ebene aller Belege und Positionen vorgenommen, wobei eine beliebige Menge dieser korrigiert werden können. Das Korrekturobjekt muss ausgewählt werden und wird in einem entsprechenden Task korrigiert. Im Anschluss folgt eine Überprüfung vorgenommener Korrekturen. Wird bei der Prüfung festgestellt, dass die Korrektur falsch oder unvollständig ist, kann zur eigentlichen Korrekturaufgabe zurückgekehrt werden. Sind alle Korrekturen abgeschlossen, folgt auf die Prüfung das Ende des Subprozesses.

Erstattungsberechtigung prüfen

Bevor eine Rechnung oder Teile einer solchen erstattet werden können, muss geprüft werden, ob überhaupt eine Erstattungsberechtigung vorliegt. Zu diesem Zweck muss der Versicherungsschutz der behandelten Person erfasst und geprüft werden. Jede

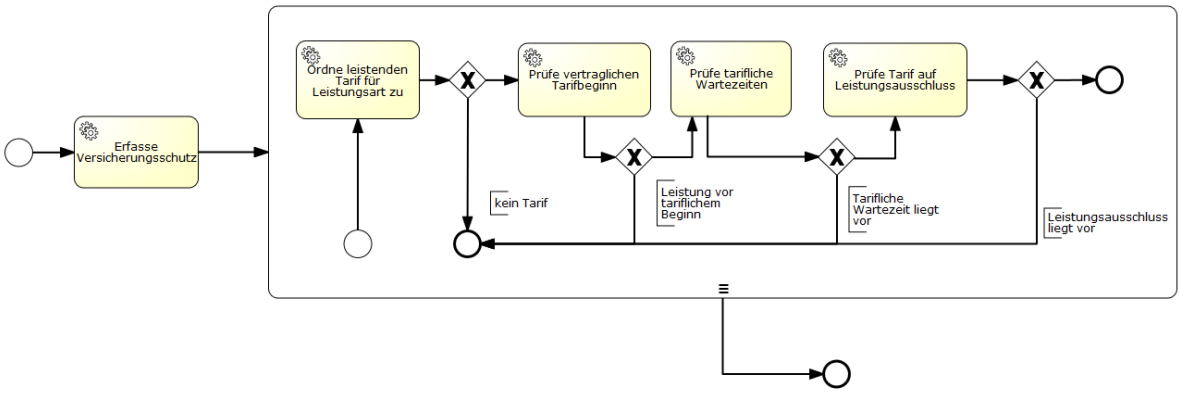


Abbildung 5 Sub-Prozess-Modell Erstattungsberechtigung prüfen

Rechnungsposition stellt eine Leistung dar und entsprechend wird für jede Position überprüft, ob ein Tarif existiert, welcher diese abdeckt. An dieser Stelle wird davon ausgegangen, dass in unserem hypothetischen Versicherungsbetrieb Leistungen nur durch jeweils einen Tarif im Vertrag abgedeckt sein können. Auf Basis des Versicherungsschutzes wird dabei die Prüfung in mehreren Schritten vorgenommen, die jeweils entsprechende Faktoren prüfen. Schlagen diese Prüfungen fehl, existiert zumindest zu diesem Zeitpunkt kein Versicherungsschutz und damit liegt keine Erstattungsberechtigung vor. Zunächst

erfolgt dabei die Prüfung, ob generell ein Tarif existiert, welcher diese Leistung abdeckt. Ist dies der Fall, wird überprüft ob die Leistung nach vertraglichem Tarifbeginn liegt, keine tariflichen Wartezeiten vorliegen und kein Leistungsauschluss vorliegt. Sind alle Prüfungen erfolgreich kann eine Erstattung der Leistung anhand des gefundenen Tarifs erfolgen.

Erstattungshöhe berechnen

Ist die Erstattungsberechtigung festgestellt wird der Erstattungsbetrag berechnet, wobei zunächst geprüft wird, ob die Rechnung bereits abgerechnet wurde. Handelt es sich um eine solche Doppelabrechnung wird die Berechnung übersprungen und der Subprozess nach Festlegung des Erstattungsbetrags aus „0“ pro Position beendet. Im normalen Verlauf erfolgt nach der Prüfung die Berechnung des Erstattungsbetrags pro Belegposition. Dabei wird zunächst der Erstattungsfähige Betrag berechnet. Dieser stellt den Betrag da, der nach prozentualem Tarifsatz erstattet wird. Er unterscheidet sich vom tatsächlichen Erstattungsbetrag durch Abzug eventueller Selbstbehalte oder anderer Faktoren, die die Erstattung reduzieren können.

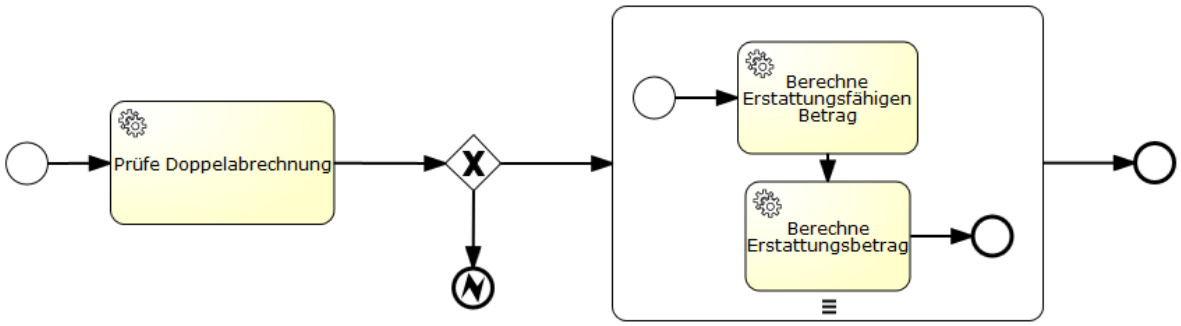


Abbildung 6 Sub-Prozess-Modell Erstattungshöhe berechnen

Manuelle Korrekturen vornehmen

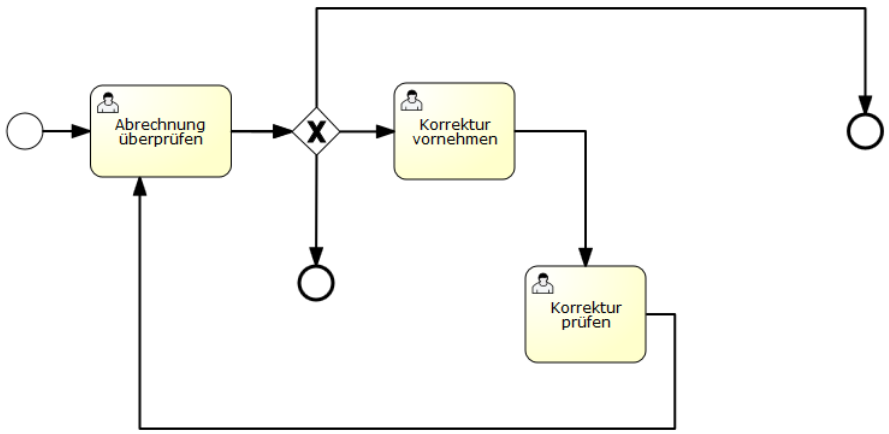


Abbildung 7 Sub-Prozess-Modell Manuelle Korrekturen vornehmen

Nachdem die automatische Prüfung und Berechnung erfolgt ist, können manuelle Korrekturen vorgenommen werden. Dabei prüft der Bearbeiter zunächst die Belege und Erstattungsbeträge. Wird in diesem Schritt festgestellt, dass ein Fehler in Belegerfassung

gemacht wurde, wird der Prozess durch eine Entscheidung in einem Exclusive Gateway mit einem entsprechenden End Event beendet. Die eigentliche Belegkorrektur erfolgt gesondert und wird im Top-Level-Prozess angestoßen. Ist kein Fehler in der Belegerfassung kann der Bearbeiter feststellen, dass eine manuelle Korrektur an den Erstattungsbeträgen vorgenommen werden soll (z.B. Kulanz). Nach dem diese ausgeführt wurde, muss eine Prüfung durch einen Vorgesetzten erfolgen. Dieser kann die Korrektur genehmigen oder rückgängig machen. Anschließend kann die Abrechnung erneut geprüft, weitere Korrekturen vorgenommen oder die Abrechnung abgeschlossen werden. Die Beendigung der Abrechnung kann auch ohne vorgenommene Korrektur erfolgen.

3.3 Zuordnung von User Tasks

Die Zuordnung von User Tasks erfolgt im Activiti BPMN Modell. Dabei wird jeder Task eines Prozesses, welcher, durch entsprechende Gruppenzugehörigkeit, vom Prozessersteller ausführbar ist, direkt diesem Benutzer zugeordnet. Im Prozess muss also eine entsprechende Prozessvariable für diese Zuordnung existieren, die dann im Modell verwendet wird. Andernfalls wird der Prozess der entsprechenden Benutzergruppe zugordnet.
Im Beispielprozess in einem hypothetischen Versicherungsunternehmen existiert die Rolle der Sachbearbeiter und der Teamleiter. Alle User Tasks sind dabei vom Sachbearbeiter ausführbar, mit Ausnahme von „Korrektur prüfen“ aus der Belegerfassung, welcher durch einen Teamleiter ausgeführt werden muss.

3.4 Tarife und Leistungen

Das System soll eine Reihe einfacher Tarife einer hypothetischen Krankenversicherung und verschiedene Leistungsarten unterstützen. Die Tarife decken mindestens eine dieser Leistungen ab.

Tabelle 6 Leistungsarten im Beispiel

Leistungsart	Beschreibung
Ambulant	Sämtliche Gebührenziffern der GoÄ werden als Leistung dieses Typs betrachtet. Weitere Leistungen außerhalb des Kataloges, die auch ambulanter Art sind, sind nicht von dieser Leistungsart abgedeckt.
Stationär	Sämtliche Krankenhausleistungen nach einem ausgedachten Katalog[14] sind stationär. Weitere stationäre Leistungen außerhalb des Katalogs sind durch die Leistungsart abgedeckt.
Zahn	Sämtliche Gebührenziffern der GoÄ. Außerhalb dieses Katelogs stehende zahnärztliche Leistungen sind nicht durch die Leistungsart abgedeckt.

[14] Der ausgedachte Katalog enthält aus den DRG abgeleitete Fälle, die wie die GoÄ und GoZ aufgebaut sind. Grund für diese Vereinfachung ist, dass die Implementierung einer Unterstützung für DRG zu komplex und nicht Gegenstand dieser Arbeit ist.

Die nicht abgedeckten, nicht katalogisierten Leistungen, die in der Tabelle beschrieben sind, können nur im Sonderfall durch Kulanz in einer Erstattung berücksichtigt werden. Bestimmte Leistungen aus diesen Leistungsarten können im Tarif ausgeschlossen werden.
Die Tarife sollen alle Konstellationen abdecken, die in Tarifen (Tabelle 7) definiert sind. Ihnen werden entsprechende Leistungen zugordnet.

Tabelle 7 Tarife im Beispiel

Tarifname	Leistungen	Bedingungen
ambulantStandard	Ambulant	100% Erstattung Bis 1000 Euro Selbstbehalt
StationLight	Stationär	90% Erstattung Kein Selbstbehalt
complete	Stationär, Zahn, Ambulant	55% Ersattung Kein Selbstbehalt
completePlus	Stationär, Zahn, Ambulant	100% Erstattung Keine Selbstbehalte

Zur weiteren Vereinfachung der Fälle, wird bei der zeitlichen Aufteilung von Leistungen aus tariflichen Gründen der Rechnungsbetrag gleichmäßig auf die Teilzeiten verteilt und nicht nach entsprechenden Details der Behandlung gegangen.

4 Anforderungen

Das Fallbeispiel aus dem vorigen Abschnitt, stellt die Anforderung an die Activiti Engine dar. Der Prozess soll in der Engine ausgeführt werden, wobei die Benutzeraktionen über eine Anwendung vorgenommen werden sollen. Diese Anwendung verwendet die Activiti Engine und stellt Oberflächen und Funktionalität zur Verfügung, um Prozesse auszuführen. Die Anforderungen an eine solche Anwendung, die sich auf die minimalen Funktionen beschränkt, sollen in diesem Abschnitt erläutert werden.

Anforderung 1: Die Anwendung soll das Benutzersystem des Activiti Frameworks einbinden, um Gruppen mit Namen und Benutzer mit Namen, Gruppenzugehörigkeit und Passwort speichern zu können.

Anforderung 2: Benutzer sollen sich über eine Maske (Log-In) mit ihren Daten anmelden können. Bei Anmeldung entsteht eine Benutzer-Session, die an die Lebensdauer der http-Verbindung gebunden ist.

Anforderung 3: Zugang zu Masken, außer der Log-In-Maske, ist nur bei Bestehen einer Session möglich. Anderweitige Zugriffsversuche werden auf die Log-In-Maske umgeleitet.

Anforderung 4: Bei erfolgreicher Anmeldung soll auf eine Übersichtsmaske navigiert werden. Diese Overview-Maske bietet eine unsortierte Übersicht, in Tabellenform, über verfügbare Prozesse und vorhandene User Tasks, die dem Benutzer zugeordnet sind.

Anforderung 5: Benutzer sollen sich auf der zentralen Overview-Maske jederzeit über einen Button abmelden können. Nach erfolgter Abmeldung wird auf die Log-In-Maske navigiert.

Anforderung 6: Die verfügbaren Prozesse können von der Overview Maske über einen Button gestartet werden. Jeder Benutzer kann dabei jeden Prozess starten. Es wird dabei immer die aktuellste Version eines Prozesses gestartet. Jeder Prozess wird dabei als Spalte mit Namen, Version und Button angezeigt.

Anforderung 7: Bei Start eines Prozesses wird nicht auf eine andere Maske navigiert. Der Benutzer erhält auf der Overview-Maske, nach betätigen des Buttons, Auskünfte darüber, ob der Prozesse gestartet werden konnte.

Anforderung 8: Tasks werden in der Overview-Maske als Tabelleneinträge mit ID-Nummer, Name des Tasks aus dem Prozessmodell, dem Prozess zu dem sie gehören mit Name und ID-Nummer, sowie ihrer Priorität als Spalten angezeigt. Eine weitere Spalte enthält einen Button, über den zur Task-Ausführung navigiert wird.

Anforderung 9: Tasks gelten dann als zu einem Benutzer gehörend, wenn diese entweder direkt zugewiesen oder über Gruppenzugehörigkeit durch den Benutzer ausgeführt werden können.

Anforderung 10: Zur Ausführung von User Tasks soll eine generische Oberfläche verwendet werden. Auf dieser Maske wird der Name des Tasks, seine Form Properties als Formular, seine Beschreibung aus dem Prozessmodell und ein Button zum Abschließen vorhanden sein.

Anforderung 11: Form Properties, die eine benötigte Eingabe darstellen, werden als Texteingabefeld angezeigt. Form Properties, die eine Auswahl zwischen Werten erfordern, werden als Combobox angezeigt, wobei nur ein Eintrag aus der Liste ausgewählt werden kann. Form Properties, die als Inhalt Prozessvariablen haben, werden als Text dargestellt.

Anforderung 12: Die Form Properties sollen sortiert nach Typ mit Namen als vorrangehender Beschreibung angezeigt werden, wobei zunächst die Ausgabe- und dann die Eingabe-Properties angezeigt werden.

Anforderung 13: Die Ausführung eines Tasks soll jederzeit durch einen Button unterbrochen werden können. Dabei wird bei Betätigung des Buttons auf die Overview-Maske navigiert.

Anforderung 14: Wird der Abschließen-Button betätigt muss geprüft werden, ob alle entsprechenden Eingaben (15) erfolgt sind. Sind alle Eingaben korrekt (17) gemacht worden, so wird der Task abgeschlossen und auf die Overview-Maske navigiert.

Anforderung 15: Form Properties, die eine Eingabe repräsentieren, sind entweder optional oder verpflichtend zu erfüllen. Ob alle verpflichtenden Auswahlen und Eingaben vorgenommen worden sind, muss in einer allgemeinen Task unabhängigen Prüfung vor Abschluss geprüft werden. Im Fehlerfall wird in Anforderung 14 verfahren.

Anforderung 16: Es muss ein fachliches Datenmodell im Kontext der Prozesse erstellt werden, welches Geschäftsprozessbestandteile und -ergebnisse darstellt.

Anforderung 17: Jeder User Task hat eine eigene Validierungsklasse, die überprüft, ob die individuellen Felder korrekt gefüllt worden sind. Dabei werden aus den Eingaben entweder direkt oder durch Konstruktion von Fachobjekten Prozessvariablen erzeugt. Sind die Eingaben nicht korrekt vorgenommen worden, wird das Abschließen abgebrochen und die Task-Maske mit einer entsprechenden Fehlermeldung angezeigt.

Anforderung 18: In einem Prozess entstandene Fachobjekte, die das Ergebnis oder seine Grundlagen repräsentieren, sind zur Laufzeit als Prozessvariablen gespeichert. Bei erfolgreichem Abschluss eines Prozesses, müssen diese Daten in entsprechende fachliche Datenbanktabellen übertragen werden.

Anforderung 19: Alle Oberflächen sollen als HTML-Seiten realisiert werden. Der Inhalt der Seiten soll durch die Anwendung generiert werden.

Anforderung 20: Die Anwendung soll in Java geschrieben werden und als Web-Application in einem Application Server laufen.

Anforderung 21: Die Activiti Engine-API muss für die Anwendung in beliebiger Form verfügbar sein.

5 Konzept [15]

5.1 Application Server

Als Application Server soll Glass Fish von Oracle genutzt werden in Version 3.1.1.2. Auf diesem soll die Webanwendung in einem *.war-Archiv über die Administrationskomponente im Browser deployt werden.

5.2 Datenbank

Für die Anwendung und für Activiti soll eine gemeinsame Postgres Datenbank in Version 8.4 eingerichtet werden. Diese Version wurde für die Verwendung mit Activiti geprüft[16]. Die Datenbank soll nur indirekt über den Application Server verfügbar sein. Es wird daher ein JDBC Connection Pool im Glass Fish eingerichtet. Diese erhält einen Namen (JNDI) und kann dann in der Anwendung als Ressource genutzt werden. Transaktionen werden über den von Glass Fish zur Verfügung gestellten TransactionManager (JTA) durchgeführt.[17]

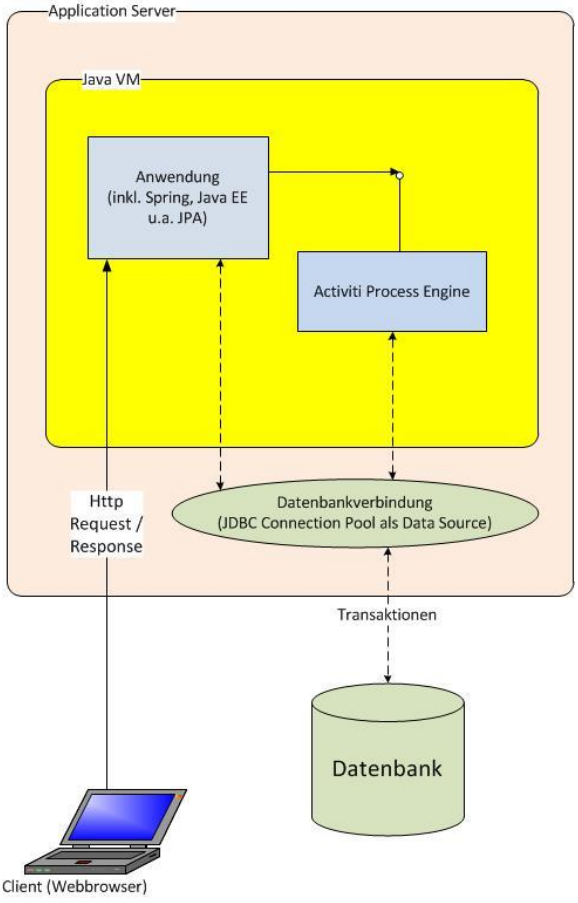


Abbildung 8 Technischer Aufbau

5.3 Activiti Process Engine

Die Activiti Process Engine wird in der Version 5.10 verwendet. Die verwendete Implementierung ist die CdiJtaProcessEngine[18] aus dem Paket org.activiti.cdi, welche von der Firma camunda zur Verfügung gestellt wird.

[15] Quelle für JEE (Java Enterprise Edition; CDI und Verwendung von Java Beans mit CDI) und Glassfish siehe [15]
[16] Siehe [10, S. 182]
[17] Weitere Informationen zu JNDI, JTA und JDBC: [16] [17] [19]
[18] Siehe Activiti User Guide [13] Abschnitt 16

Datenbankanbindung und -transaktionen erfolgen mit Hilfe der von Glass Fish zur Verfügung gestellten Ressourcen. Es wird das Historisierungslevel „full“ verwendet, um komplette Historisierung (Anforderung 4) zu gewährleisten.
Die verwendete Implementierung bietet die Möglichkeit Named Beans im Prozessmodell zu verwenden, um entsprechende Beans oder Methoden aus diesen Beans bei der Ausführung von Service Tasks aufzurufen.
Die Process Engine selber soll vom Application Server als Service über den java.util.ServiceLoader verwaltet werden. Dadurch können die Schnittstellen zu den APIs (TaskService etc.) der Engine in der gesamten Applikation per Dependency Injection injiziert werden.

5.4 Kapselung der Engine Zugriffe

Sämtliche Zugriffe auf die Process Engine sollen über eine Klasse ActitivitiEngineServices gekapselt werden. Dieser Klasse werden alle benötigten APIs wie in 2.2.10 - 2.2.15 beschrieben verfügbar gemacht. Erweiterte Funktionalität der Process Engine kann so zu Methodenaufrufen gebündelt und in der Anwendung genutzt werden.

5.5 Webanwendung

Die Webanwendung wird mit Maven Archetype für Webapplications erstellt und über ein entsprechendes Maven-Goal wird die Webanwendung als .WAR-Datei gebaut. Mit dem Maven Archetype wird im Projekt eine Webanwendungsstruktur mit benötigten Verzeichnissen generiert. Alle Abhängigkeiten werden über das Maven Project Object Model (POM) verwaltet.

5.6 Benutzer Session

Eine Benutzer Session beginnt nach dem Login und endet bei Beendigung der http-Session zum Client. Eine Session soll daher mit einer CDI-Bean (Bestandteil Java EE ab Version 6) realisiert werden, welche einen Session Scope besitzt. Diese Beans werden vom Application Server (siehe 5.1) verwaltet und können durch Dependency Injection an den entsprechenden Stellen verfügbar gemacht werden. Diese Funktionalität ist Standard für CDI-Bean in Java EE ab Version 6.

5.7 Session Daten

Die Bean zur Repräsentation der Session benötigt Daten über den jeweiligen Benutzer und den Zeitpunkt der Erstellung der Session. Als Benutzerdaten ist der Benutzername ausreichend. Diese Daten müssen für eine gültige Session entsprechend gefüllt werden.

5.8 User als Prozessvariable

Beim Starten eines jeden Prozesses wird der Benutzername als Prozessvariable gesetzt. So können alle Tasks aus der entsprechenden Berechtigungsgruppe automatisch diesem Benutzer über den Task zugeordnet werden. Zum Setzen der Variable wird ein Listener verwendet, welcher beim Start Event ausgelöst wird.

5.9 Oberflächen

Oberflächen sollen mit Servlets umgesetzt werden, wobei die eigentlichen Oberflächen durch direkten Output dieser erzeugt werden, ohne Verwendung von XHTML-, JSP- oder JSF-Seiten. Im Output befindet sich der HTML-Code, der die eigentliche Seite darstellt. Da ein solches Servlet damit Logik zur Verarbeitung von Eingabe und reine Ausgabefunktion hat, soll die Ausgabefunktionalität in eine Klasse pro Servlet ausgelagert werden. Diese Klasse trägt denselben Namen wie das Servlet, allerdings mit dem Suffix „Print“.
Die Navigation von Oberfläche zu Oberfläche wird in den Servlets, aus denen heraus auf ein anderes Servlet navigiert wird, bestimmt.

5.10 Servlet Klasse

Es wird eine Erweiterung der Klasse HttpServlet[19] aus dem Paket javax.servlet.http verwendet. Das HttpServlet bietet die Zugriffsmethoden post und get entsprechend dem http (siehe [3]). Entsprechende Methoden können in erbenden Klassen mit spezifischer Funktionalität überschrieben werden. Die Methoden erhalten dafür die entsprechende Request und die Response (nach http siehe auch [4]). Der Request enthält entsprechende Variabeln, die zum größten Teil über Formularelemente oder auch direkt über das Objekt in einem Servlet gesetzt werden. Die Response verfügt über einen PrintWriter um Output zu schreiben. Diesem PrintWriter wird html-Code in Strings an seine println-Methode übergeben.

5.11 Session Prüfung im Servlet

Da Funktionalität nur dann verfügbar ist, wenn eine gültige Session vorliegt, muss eine entsprechende Prüfung durchgeführt werden. Die Erweiterung der Klasse HttpServlet, SessionCheckedHttpServlet, bietet Methoden zur Prüfung der Session Bean, welche in den entsprechenden get- oder post-Methoden aufgerufen werden müssen.

Servlet Klassen Hierarchie:

HttpServlet *(Implementierung des Interfaces javax.servlet.Servlet)*
- ➔ SessionCheckedHttpServlet
  - ➔ Overview, Task, Login

[19] Dokumentation Klasse HttpServlet siehe [20]

5.12 Logout

Es soll eine Klasse SessionCommonsPrint erstellt werden, in der eine zentrale Methode Output generiert, der den Namen des aktuellen Benutzers anzeigt und einen Button für den Logout bereitstellt. Nach betätigen dieses Buttons wird auf die Login Seite navigiert.

5.13 Fachliches Datenmodell

Eine Leistungsabrechnung ist einem Vertrag zu zuordnen. Sie besteht aus mindestens einer abzurechnenden Rechnung, die wiederrum aus jeweils mindestens einer Rechnungsposition besteht, welche im Verlauf der Abrechnung in verschiedene Abrechnungsteile unterteilt wird. Rechnungen gehören zu genau einer Versicherten Person des Vertrages, welche wiederrum einer natürlichen Person entspricht. Jede Versicherte Person hat mindestens einen Tarif, wobei jeder Tarif mindestens eine Leistungsart abdeckt. Für den konkreten Tarif, auch Tarifinstanz genannt, können je nach Situation Leistungsausschlüsse und Wartezeiten vereinbart sein. Des Weiteren hat jeder Vertrag genau einen Versicherungsnehmer.
Die Primärschlüssel werden durch Inkrement ab dem Wert „1“ pro Entität vergeben. An der Spitze steht dabei der Wert für die Leistungsabrechnung, bei der dieser Wert alleinig den Schlüssel definiert. Abhängige Entitäten erhalten zusätzlich den Primärschlüssel der übergeordneten Entität als Fremdschlüssel in einem zusammengesetzten Primärschlüssel, wobei das Inkrementieren pro übergeordneter Entität erfolgt.

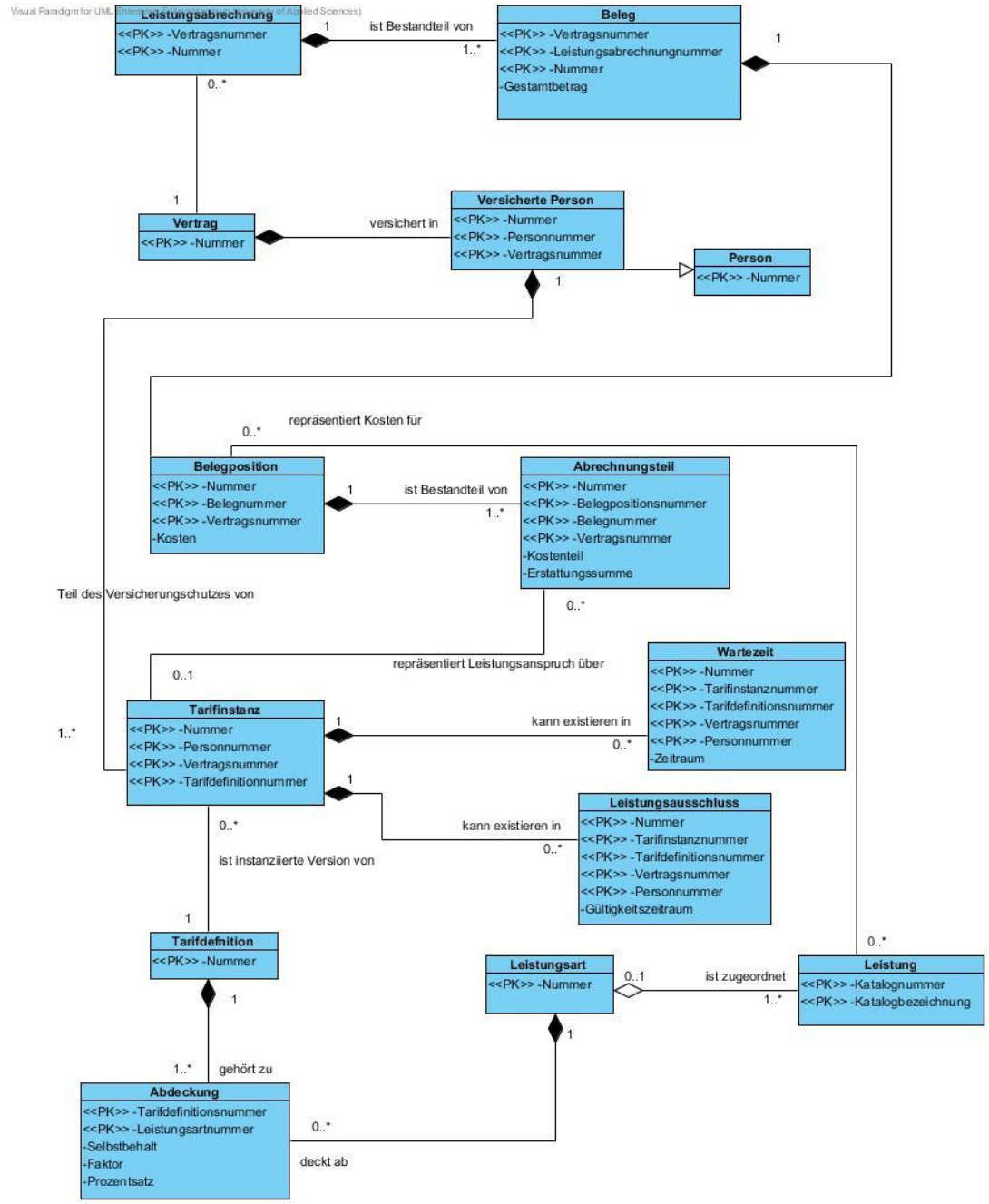


Abbildung 9 Fachliches Datenmodell

5.14 Persistierungszeitpunkt

Die Persistierung von Fachobjekten erfolgt stets nach Abschluss des jeweiligen Tasks. Eine Speicherung während der Prozesslaufzeit erfolgt in Form von Prozessvariablen.

5.15 Prozessabschluss

Über einen Listener auf dem End Event, welches das erfolgreiche Prozessende darstellt, werden die Daten in den entsprechenden Tabellen gespeichert. In nicht erfolgreichen Prozessenden werden keine Daten persistiert. Allerdings sollten die in der Datenbank gespeicherten Prozessdaten für solche Prozesse regelmäßig gelöscht werden, um die Größe der Datenbank und damit die Performance der Engine nicht zu gefährden.

5.16 JPA[20]

Glass Fish verfügt über die JPA Implementierung EclipseLink, welche für Datenbankzugriffe, schreibend und lesend, verwendet werden soll. Diese Transaktionen sollen über den Application Server (siehe 5.1) durchgeführt werden. Eine entsprechende Persistence Unit wird erstellt, in der alle Fachobjektklassen, sowie technische Objektklassen referenziert sind.

5.17 Prozessvariablen

Die Bezeichnungen für die Prozessvariablen sollen in einer Klasse „*Constants*“ als statische Felder abgelegt werden. Damit soll die Verwendung von falschen Bezeichnern verhindert und eine zentrale Stelle für die Verwaltung eingerichtet werden. So kann verhindert werden, dass bereits bestehende Variablenbezeichnungen neu definiert werden oder mögliche Änderungen der Bezeichner zentral erfolgen.

5.18 User Tasks

5.18.1 Übergabe der Task-ID

Die Task-ID ist in einer Variablen im Request-Objekt gespeichert. Das Task-Servlet besitzt einen Zustand, der den Zustand der Verarbeitung repräsentiert. Diese Zustände sind:

- INITIAL*: Anfangszustand bei Zugriff, Prüfe Session*
- PROCESSINPUT: *Formular für Prozessausführung wird ausgewertet*
- COMPLETETASK: *Task hat den benötigten Input und die Verarbeitung ist abgeschlossen; Task wird abgeschlossen*

5.18.2 Darstellung

Die generierte Oberfläche pro Task wird jeweils mit einer Überschrift versehen, die dem Tasknamen aus dem BPMN Modell entspricht. Die Unterscheidung zwischen Out- und Input erfolgt über das Value-Attribut der Form Properties. Ist dieser Wert gesetzt, handelt es sich um eine Prozessvariable, welche automatisch durch Activiti ergänzt wurde und die als Text ausgegeben wird. Ist kein Wert gesetzt handelt sich um eine Eingabefeld oder eine

[20] Java Persistence API; weitere Informationen siehe [18]

Auswahlmöglichkeit, die erzeugt werden muss. Dieser wird mit Namen und Textfeld bzw. Auswahlfeld in einem Formular ausgegeben. Das Formular wird per Post an dasselbe Servlet übergeben. Vor Generierung der Ausgabe wird die Liste der Form Properties nach dem Value-Attribut sortiert.
Form Properties und Taskinstanz werden durch gekapselte Engine Zugriffe ermittelt.

5.18.3 Task spezifische Elemente

Die Task spezifischen Elemente werden mit Spring Beans realisiert. Im Task Servlet wird entsprechend dem Taskbezeichner aus dem BPMN Modell, der Name der entsprechenden Bean aus der Datenbank ermittelt. Dazu wird eine technische Entität erstellt, die eine solche Zuordnung vornimmt.

5.19 Overview

Die Funktionalität Prozesse zu starten oder User Tasks auszuführen wird über Parameter gesteuert.

5.19.1 Prozessinstanzen starten

Über die gekapselte Engine Schnittstelle werden alle verfügbaren Prozessdefinitionen geladen und entsprechend mit Prozessnamen ausgegeben. Über ein Post-Formular kann ein Prozessstart ausgelöst werden. Dazu wird ein Button verwendet und ein Hidden Field mit Namen der Prozessdefinition. Ein Prozessstart wird dann entsprechend über die Engine ausgeführt.

5.19.2 User Tasks

Es werden alle User Tasks mit entsprechenden Attributen und entsprechend, ebenfalls mit Formularen und Hidden Fields, kann die Ausführung des Tasks angefordert werden. Dazu wird der http-Request mit entsprechenden Variablen versehen und eine Weiterleitung zum Task Servlet vorgenommen.

5.20 Fachlicher Komponentenschnitt

In einer klassischen fachlichen Aufteilung würde die Anwendung zur Realisierung z.B. eine Leistungssystem-, Partnersystem- und eine Vertragssystemkomponente besitzen. Die benötigte Funktionalität beschränkt sich dabei auf das Laden und Schreiben von Daten nach fachlichen Kriterien, wie einer Versicherungsnummer. Um diesen kleinen Ausschnitt der Funktionalität anbieten zu können, werden die Partnersystem- und Vertragssystemkomponente ganz, die Leistungssystemkomponente teilweise (Softwareartefakte des Prozesses zählen ebenfalls zur Komponente) durch je eine entsprechend benannte Klasse repräsentiert.

6 Umsetzung

Die Umsetzung des Anwendungskonzepts ist ohne Abweichung vorgenommen wurden. Im nachfolgenden Kapitel soll sowohl diese Umsetzung behandelt werden als auch das Fallbeispiel anhand seiner Gestaltung ausgewertet werden.

6.1 Verwendete BPMN Elemente

Der Prozess Leistungsabrechnung, welcher als Fallbeispiel dient, enthält bei weitem nicht das volle Spektrum an BPMN Elementen. Dies liegt zum größten Teil an der fachlichen Vorgabe, aber auch an der Verknüpfung mancher Elemente mit weiteren Frameworks (z.B. Drools für Business Rule Tasks) und auch an der Tatsache, dass in der aktuellen Version 5.11 nicht alle Elemente von Activiti unterstützt werden.

Thema dieser Arbeit ist nicht die Abdeckung sämtlicher Arten von BPMN Elementen. Es soll lediglich eine grundlegende Menge an Konzepten abgedeckt werden. Diese werden als grundlegend betrachtet, da weitere Elemente ähnlicher Art lediglich eine Verfeinerung des Konzepts darstellen oder sie als Elemente nicht in vielen Prozessen oder nur sehr (technisch) verfeinerten Prozessmodellen Anwendung finden. Ein Beispiel für Elemente der letzteren Art sind Cancel Events und Compensation. Eine Auflistung der Funktionen und der damit verbundenen Elemente kann folgender Tabelle entnommen werden:

Tabelle 8 BPMN Konzepte & Elemente

Konzept	Elemente
Einfacher und explizit konditionaler Workflow: Es bestehen mehrere, verschiedene potentielle Pfade durch den Prozess, aber stets nur ein einspuriger Pfad in einem Prozessverlauf.	Sequence Flow in Verbindung mit Explicit Gateways als Gabelungen / Zusammenführungen.
Konditionaler und nicht konditionaler paralleler Workflow: Es bestehen mehrere, verschiedene potentielle Pfade durch den Prozess, aber stets nur ein ein- oder mehrspuriger Pfad in einem Prozessverlauf.	Sequence Flow in Verbindung mit Implicit, Event (konditional) und Parallel Gateway (nicht konditional). Es bestehen mehrere verschiedene Pfade und gleichzeitig ein- oder mehrspurige Verläufe durch einen Prozess.
Manuelle Aufgaben, die nur durch den Benutzer bearbeitet werden können[21]: Der Benutzer bearbeitet seine Aufgabe und gibt Daten ins System ein.	User Tasks

[21] Ausnahme: Verarbeitung der User Eingaben durch eine Maschine

Untergeordnete Prozesse inner- oder außerhalb des Prozessrahmens: Ermöglichen die Gruppierung und hierarchische Anordnung von Aufgaben in Teilaufgaben aus mehreren Ebenen.	Subprocess: Teilprozess im Prozessrahmen Call Activity: Teilprozess außerhalb des Prozessrahmens
Mehrfachausführung von Aufgaben: Sequentielle oder parallele Mehrfachausführung von Aufgaben und Untergeordnete Prozesse	Multi-Instance parallel oder sequentiell
Automatische Aufgaben, die ausschließlich von einer Maschine ausgeführt werden: Die Aufgabe greift auf Informationen im System zu und arbeitet auf Basis dieser Daten bzw. bearbeitet diese Daten.	Service Task als grundlegendes Element. Weitere Elemente sind u.a. Business Rule Tasks und Web Service Tasks. Diese Elemente sind nur Verfeinerungen des Service Tasks, da sie letztendlich dasselbe Konzept darstellen, nur mit weiteren Spezifikationen. Beispiel: Web Services definieren Service als Web Service mit entsprechender wsdl-Beschreibung; Business Rule Task: Service definiert als eine Reihe automatischer Regelprüfungen.
Im Workflow oder an Aufgaben gebundene Ereignisse: Ereignisse, die abgefangen und/ oder ausgelöst werden können. Das Auftreten und abfangen von Ereignissen kann an den Rahmen einer Aufgabe gebunden sein (können auftreten) oder direkt im Workflow explizit auftreten (müssen auftreten). Bestimmte verbundene Ereignisse in verschiedenen Prozessen modellieren Interprozesskommunikation.	• Error Events zur Darstellung fachlicher Fehler • Message Events zur Darstellung gesendeter und empfangener Nachrichten von außerhalb des Prozesses liegenden Entitäten • Signal Events zur Darstellung globaler Signale • Timer Events zur Darstellung von Zeitpunkten und Zeiträumen • Weiterhin: Event Subprozesse. Sie definieren den Teilprozess als Verarbeitungsrahmen in dem ein Ereignis auftritt. Sonst ist es der übergeordnete Prozess.

Von den wesentlichen Elementen fehlt vor allem der parallele Workflow in Verbindung mit entsprechenden Gateways für Gabelungen und Zusammenführung. Da dieser aus fachlichen Gründen an keiner Stelle sinnvoll war, soll er in weiteren Beispielprozessen, im Verlauf des Kapitels, verwendet werden. Dasselbe gilt für Message- und Timer Start Event.

Als technische Hilfe kann weiterhin der Script Task bezeichnet werden. Dieser Task Typ ermöglicht die Ausführung von in Script geschriebenem Code. Dabei kann dieser Code, in Activiti, in vielen Script Sprachen geschrieben werden, wie z.B. Groovy oder Java Script. Script Tasks sollten verwendet werden für das Setzen und einfache (z.B. arithmetische) Manipulieren von Prozessvariablen, welche eine wichtige Rolle in der Prozesssteuerung

spielen. Ein Beispiel findet sich in Abschnitt 1.2.1., in dem Script Tasks zum Setzen von Zähler-Variablen verwendet werden.

6.2 Daten und Dokumente

Die bisher erwähnten Konzepte und damit verbundenen Elemente beschreiben einen Prozess in seiner Ablaufstruktur mit verschiedenen Aufgabentypen und allen Ereignissen, die im Prozessverlauf auftreten können. Ein Prozess, gerade auf der Workflow Ebene, benötigt Zugriff auf bestehende Daten und erstellt und manipuliert seinerseits wiederum Daten. Es erfolgt dabei im Modell meist keine explizite Darstellung der benötigten (Input) und der erzeugten oder manipulierten (Ouput) Daten an entsprechenden Tasks, sondern es wird implizit davon ausgegangen, dass ein Prozess bzw. die Processengine auf diese Daten als Prozessvariablen zugreift und diese verwaltet. BPMN kennt weiterhin Data Stores, um bestimmte Datenpools (z.B. eine Partner- oder Belegdatenbank) darzustellen. Damit wird festgelegt und veranschaulicht aus welchem bzw. in welchen Datenpool Daten gelesen bzw. geschrieben werden. Auf dieses Element wird jedoch verzichtet, da in dem System lediglich ein zentraler Datenpool für alle Zwecke verwendet wird und, durch fehlenden Support dieses Elements durch die Process Engine, dieses Element lediglich dokumentierenden Charakter hätte.

Dokumente, welche in BPMN 1.X als Annotations im Dokument ohne semantische Bedeutung waren, werden in BPMN 2.0 als vollwertige Elemente betrachtet. Dokumente können in Geschäftsprozessen wichtige Elemente darstellen. Aus Gründen der Komplexität, der technischen Möglichkeiten der verwendeten Engine wird im gesamten Prozess auf Dokumente verzichtet. Ein Message Event ist daher in der Anwendung lediglich ein Signal, welches die Ankunft eines Dokuments darstellt.

6.3 Bewertung der Umsetzung von BPMN Elementen

Die Umsetzung der Elemente soll anhand ihrer Spezifikation bewertet werden. Dabei wird zunächst ein Beispielprozess erläutert, welcher weitere Elemente und Konzepte umsetzt. Anschließend wird eine Bewertung vorgenommen. Dabei werden Details, wie die Vorgabe von Zeitformaten nach ISO 8601-Standard, nicht berücksichtigt, sondern nur einschränkende Probleme aufgeführt.

6.3.1 Zusätzlicher Beispielprozess

Abbildung 10 zeigt zwei abstrakte Beispielprozesse, ohne fachlich zu Grunde liegende Prozesse.
Prozess 1 besteht aus einem Task mit anschließendem Throwing Message Intermediate Event und wird nach einer zeitlichen Vorgabe gestartet. Die Message ist Auslöser von Prozess 2, welcher entsprechend über ein Message Start Event gestartet wird. Prozess 2 besteht aus unkonditional parallelem Sequence Flow, welcher durch ein Parallel Gateway auseinander geht und in einem anderen Parallel Gateway wieder zusammengeführt wird. In einem Sequence Flow wird zunächst ein Signal Intermediate Event ausgelöst und anschließend ein Task ausgeführt. Im anderen Sequence Flow wird auf ein entsprechendes Signal gewartet und anschließend ebenfalls ein Task ausgeführt. Dabei handelt es sich in

beiden Fällen um automatische Tasks. Die Pfade werden nach der Ausführung in einem Parallel Gateway zusammengeführt und der Prozess endet.

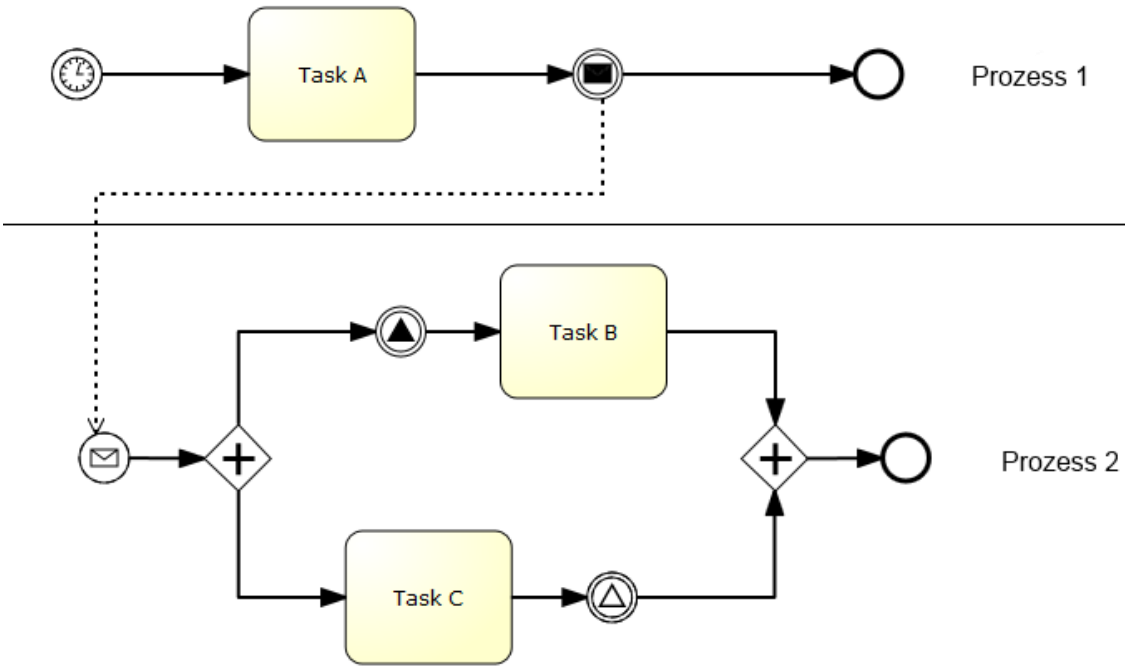


Abbildung 10 Weitere Beispielprozesse

6.3.2 Mehrfachausführung von Call Activities

Die Mehrfachausführung von Call Activities ist nur beschränkt möglich. Die Konfiguration von Multi-Instanz-Aktivitäten ist sowohl mit Activiti spezifischen Erweiterungen als auch durch BPMN-Standard auf Basis drei verschiedener Ansätze möglich:

- Iteration über eine Liste von Elementen
- Statische / dynamische Anzahl an Wiederholungen
- Bis zur Erfüllung einer Logischen Bedingung

Im Fallbeispiel soll über eine Liste von Belegen bei der Erstattungsprüfung iteriert werden. In der Call Activity sind Ein- und Ausgabeparameter definiert. Die Iteration und das Auslesen der Parameter erfolgt fehlerfrei, allerdings erfolgt keine Rückgabe der Ergebnisse an den aufrufenden Prozess. Dies lässt sich anhand der darauf folgenden Call Activity feststellen, in der diese als Input verwendet werden. Durch Ausprobieren der in der Aufzählung genannten Möglichkeiten in Kombination mit verschiedenen Folgeaktivitäten, konnte das Problem eindeutig mit der beschriebenen Kombination aus Multi Instanz und Call Activity verbunden werden. In BPMN lässt sich als Workaround-Lösung eine Call Activity in Verbindungen mit einem Exclusive Gateway verwenden. Es wird ein Zähler als Prozessvariable implementiert, der bei jeder Ausführung der Call Activity hochgezählt und im Gateway gegen die Anzahl der vorhandenen Belege geprüft wird. Ein Sequence Flow ist dabei mit der Call Activity und einer mit der weiterführenden Task verbunden.

```
<callActivity id="erstattungBerechnen" name="Erstattungshöhe Berechnen"
calledElement="erstattungBerechnen">
  <extensionElements>
          <activiti:in source="user" target="user"></activiti:in>
          . . .
          <activiti:out source="coverages" target="coverages"></activiti:out>
  </extensionElements>
</callActivity>
<boundaryEvent id="boundaryerror3" name="Error" attachedToRef="erstattungBerechnen">
          <errorEventDefinition errorRef="double"></errorEventDefinition>
</boundaryEvent>
<exclusiveGateway id="repeatBerechnung" name="weitereErstattungsberechnung" />
. . .

<sequenceFlow id="flow4" sourceRef="erstattungBerechnen" targetRef="repeatBerechnung" />
<sequenceFlow id="flow17" sourceRef="boundaryerror3" targetRef="repeatBerechnung“ />
<sequenceFlow id="flow15" sourceRef="repeatBerechnung" targetRef="erstattungBerechnen">
  <conditionExpression xsi:type="tFormalExpression">
    <![CDATA[${countBillsCalc<amountBills}]]>
  </conditionExpression>
</sequenceFlow>
<sequenceFlow id="flow16" sourceRef="repeatBerechnung" targetRef="manuelleKorrektur">
  <conditionExpression xsi:type="tFormalExpression">
  <![CDATA[${countBillsCalc>=amountBills}]]></conditionExpression>
</sequenceFlow>
```

XML-Code der Workaround Lösung für Mehrfachausführung der Call Activity „erstattungBerechnen“. Die Variable „amountBills“ repräsentiert die Anzahl der bereits verarbeiteten Belege.

Die Initialisierung der Zähler und Grenzwerte erfolgt in einem Script Task. Dadurch ist sichergestellt, dass diese Werte an einer zentralen Stelle gesetzt werden, welche sowohl unabhängig von den Software Elementen als auch für alle Betrachter des Prozessmodells sichtbar ist.

6.3.3 Paralleler Workflow

Bei der Ausführung der Prozesskombination aus dem Beispielmodell (siehe Abbildung 10) zeigt sich, dass trotz Platzierung des Catching Signal Events in der Reihenfolge vor dem Throwing Signal Event, das Catching Signal Event nicht ausgelöst wird. Grund dieses Verhaltens ist die Tatsache, dass die Ausführung des Workflows pseudo-parallel erfolgt. Die Engine bündelt alle automatischen Elemente zwischen zwei Wartezuständen (z.B. Gateways, User Tasks) zu einem einzigen, am Stück ausgeführten, Strang von Aktivitäten. Diese Stränge werden in Activiti Jobs[22] genannt, in jeweils einem Thread ausgeführt und bilden eine einzelne Transaktion. Zwischen den beiden Gateways werden also zuerst der obere und dann der untere Sequence Flow gesamt ausgeführt. Somit entsteht ein dauerhafter Wartezustand im unteren Pfad und der Prozess kann nicht abgeschlossen werden[23]. Wäre Task B keine automatische Aktivität würde dieses Problem nicht auftreten.
In einem strukturell gleichen Fall in einem nicht fiktiven Geschäftsprozess muss diese Tatsache berücksichtigt werden, wenn eine Abhängigkeit zwischen den Pfaden besteht.

[22] Siehe auch [10, S. 384-386]
[23] Da es sich in dem Beispiel um ein Signal Event handelt, würde der Wartezustand beim Starten einer neuen Prozessinstanz verlassen werden, da Signal Events global gelten.

Dabei muss es sich nicht um ein Signal Event handeln, es kann auch eine Abhängigkeit zu erzeugten Daten bestehen. Um sicherzustellen, dass der Pfad, der die Abhängigkeit besitzt, zuerst ausgeführt wird, muss dieser im XML-Modell vor dem zweiten Pfad definiert werden. Davor definiert bedeutet in diesem Fall nur, dass das erste definierte Sequence Flow Element definiert, welcher Pfad ausgeführt wird[24]. Eine weitere Möglichkeit besteht darin automatische Aktivitäten zur asynchronen Ausführung mit dem Attribut „activiti-async=true“ zu markieren, wodurch die Engine die markierte Aktivität in einem eigenen Thread ausführt und als einzelne Transaktion betrachtet. Im Beispiel würde somit ein eigener Job für die Ausführung von Task B entstehen, nach dessen Fertigstellung der untere Sequence Flow ausgeführt wird. Nach dem dieser durch das Catching Event einen Wartezustand erreicht hat, wird der obere Pfad fortgesetzt und beide Events werden ausgelöst.

6.3.4 Message Intermediate Throwing Event

In Activiti wird das Senden von Nachrichten (Messages) derzeit nur über Methoden im RuntimeService unterstützt[25]. Eine Implementierung in Form von Events ist nach Aussagen von Entwicklern nicht geplant, da die Funktionalität über die API der Engine bereits gegeben ist[26]. Diese spezifische Entscheidung und Implementierung macht es erforderlich, das Event durch z.B. einen Service Task zu ersetzen und schwächt die Bedeutung und Lesbarkeit des BPMN-Modells, da das Senden innerhalb der Softwareartefakte geschieht.

6.4 Funktionale Erweiterungen

Aus den Anforderungen und dem daraus resultierenden Konzept, gehen im Besonderen zwei Schwachpunkte hervor, welche sich besonders nach der Umsetzung deutlich hervor getan haben:

- Diskontinuität des Arbeitsablaufs
- Beschränkung von User Tasks auf eine einzige, generische, Oberfläche

Für beide Probleme werden eine genaue Problembeschreibung, ein Konzept und eine entsprechende Implementierung der Lösung vorgenommen.

6.4.1 Kontinuierlicher Arbeitsablauf

Die Belegerfassung der Leistungsabrechnung ist ein Teilprozess, welcher aus mehreren Schritten und möglichen Wiederholungen bestimmter Pfade oder Schritte besteht, die durch denselben Bearbeiter ausgeführt werden. Eine Änderung, wie das Umschreiben von Prozessen auf andere Bearbeiter, ist zwar möglich und unter den Gesichtspunkten von Urlauben oder Krankheiten, welche die Bearbeitung eines Leistungsantrags stark verzögern

[24] Die Bestätigung, dass der JobExecutor Jobs pseudo-parallel hinführt und der Hinweis, dass die Reihenfolgenvertauschung in diesem Fall das Problem lösen kann, stammt vom Activiti Entwickler Joram Barrez (siehe auch http://forums.activiti.org/comment/13773#comment-13773).
[25] Siehe [13] Abschnitt „Message Event Definition“
[26] Aussage vom 09.06.2011, siehe http://forums.activiti.org/comment/7461#comment-7461

können, sinnvoll, aber nicht der Normalfall. Daher ist es in jedem Fall effektiver einen direkt auf einen anderen User Task folgenden User Task, welcher dem Bearbeiter zugeordnet ist, direkt anzusteuern, anstatt auf dem Umweg über die Overview-Maske zu dessen Ausführung zu gelangen.
Im vorliegenden Szenario ist der Fall eines einfachen, explizit konditionalen Workflows gegeben, so dass zu jedem Zeitpunkt der Ausführung genau ein User Task vorliegen kann. Es können allerdings alle der folgenden Fälle in Prozessen vorliegen:

- Auf einen User Task folgt genau ein weiterer User Task bei parallel vorliegenden User Tasks.
- Auf einen User Task folgen im Verlauf 2 oder mehr weitere User Tasks, ohne weitere parallele Pfade mit User Tasks.
- Aus einem User Task folgen 2 oder mehr User Tasks, während parallel ebenfalls User Tasks vorliegen.

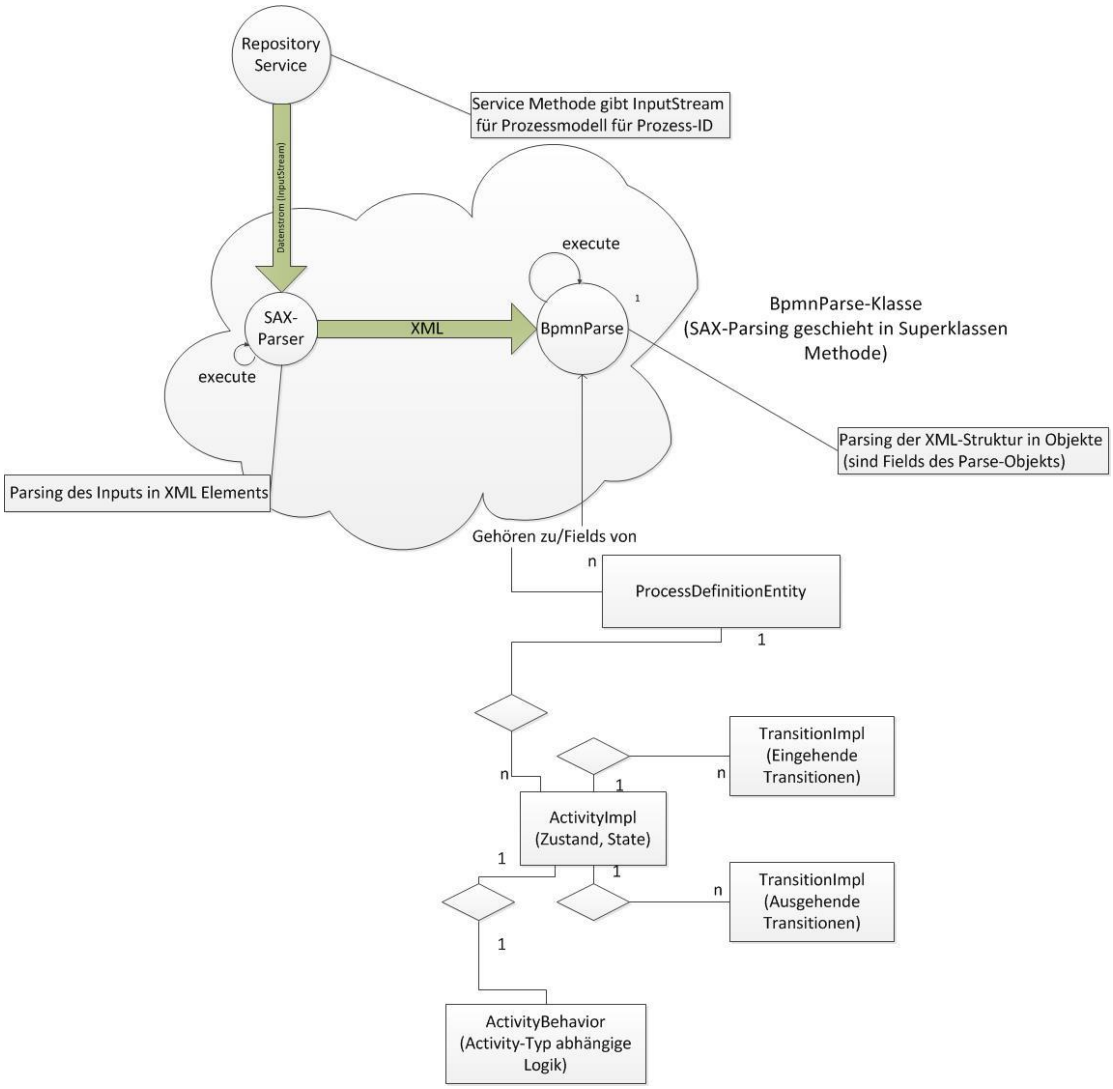


Abbildung 11 Veranschaulichung des BPMN-Parsings
Veranschaulichung des Vorgangs: Datenstrom der Quelle wird in Prozess repräsentierenden Zustandsautomaten geparst. Auftretende Fehler werden in der Grafik nicht berücksichtigt.

Da beliebige Prozesse in der Anwendung ausgeführt werden können, muss es auch möglich sein, auf letzteren Fall korrekt zu reagieren. Die Anforderung dazu lautet:

Liegt ein Nachfolge-Task vor, führe diesen aus. Liegen mehrere Nachfolge-Tasks vor, navigiere zur Overview und zeige diese einmalig priorisiert und farblich hervorgehoben an.

Mit der von Activiti zur Verfügung gestellten API lässt sich zu einem Zeitpunkt feststellen, welche aktiven Tasks im Prozess vorliegen, allerdings kann nicht von einem bestehenden Task auf seine Vorgänger, noch auf seine potentiellen Nachfolger nach der Prozessdefinition geschlossen werden, da diese Transitionen zur Laufzeit innerhalb der Engine erfolgen. Es werden dazu auch keine Informationen zu diesem Verlauf in der Datenbank gespeichert, so dass auch nicht mit einem vorgegebenen oder einem *native (SQL-Anfrage als String)* Query auf diese Informationen mit der API zugegriffen werden kann. Dies gilt ebenfalls für Anfragen auf historische User Tasks.

Innerhalb der Engine existiert ein, auf einem SAX-Parser (*Simple API for XML*) basierender, BPMN Prozess Parser. Dieser Parser wird von der Engine primär für die Verarbeitung von Deployments verwendet, die aus dem XML-Modell Objekte für die Ausführung generiert. Es lassen sich damit aber auch die zur Laufzeit die zur Ausführung des Prozesses verwendeten Objekte erzeugen. Die Objekte, welche die Elemente (Taks und Events) darstellen, sind, vernachlässigt man die Logikkomponente in den ActivityBehavior-Entitäten der jeweiligen Elemente, Knoten in einem gerichteten Graphen, denen ihre eingehenden und ausgehenden Kanten (Sequence Flow-Objekte) zugeordnet werden. Weiterhin ist dieses Objekt-Konstrukt auch als ein Zustandsautomat interpretierbar, in dem der Zustand durch die Menge der aktiven Elemente repräsentiert und das allgemeine Verhalten von Elementtypen durch die genannten Logikkomponenten bestimmt wird. Die Knoten und Kanten können dabei eindeutig über das Attribut „id“ ihres jeweiligen XML-Elements identifiziert werden. Auf der Suche nach nachfolgenden User Tasks durch den Teilgraphen können mehrere Zwischenknoten zwischen dem User Task und seinen Nachfolgern liegen. User Task-Instanzen können durch ihre Implementierung der ActivityBehavior als solche mit einer If-Kontrollstruktur identifiziert werden. Die Implementierung für User Tasks ist *UserTaskActivityBehavior*.

Es lässt sich allerdings nicht feststellen, ob weitere manuelle Aktivitäten, z.B. ein Intermediate Message Catch Event oder andere, zwischen dem Task und seinem Nachfolger liegen, daher wird in umgekehrte Richtung nach folgender Methodik ermittelt:

a) Merke die Task-ID vor Abschluss des Tasks
b) Speichere die Menge an vorhandenen Tasks, die für den Bearbeiter verfügbar sind außer dem abzuschließenden Task
c) Schließe den Task ab
d) Ermittele die Menge der Tasks für den Bearbeiter (ohne die vorher vorhandenen Tasks) nach Abschluss der automatischen Verarbeitung
e) Beschränke die Menge der Tasks auf die Tasks die den abgeschlossenen Tasks als Vorgänger haben
f) Ist nur ein Task vorhanden, führe diesen aus
g) Sind mehrere Tasks vorhanden, verzweige auf die Overview Maske und hebe diese Tasks als Gruppe hervor

Diese Logik ersetzt den bisherigen User Task-Abschluss in der Taskkomponente. Implementiert wird dazu die Methodensignatur *List<String> completeUserTaskWithSucessorDetermination(String taskId)* im Interface EngineServices. Eine entsprechende Implementierung der Punkte a) bis e) wird in der Activiti spezifischen Implementierung vorgenommen. Die Tasks werden dabei über ihre ID in einem String repräsentiert. Im Falle mehrerer vorhandener User Tasks wird ein Argument „PrioritisedTasks“ in der Navigation übergeben, welches entsprechende Task-IDs enthält. Diese werden in der Liste der Tasks, gruppiert an erster Stelle in einer anderen Farbe hervorgehoben, angezeigt.

6.4.2 Erweiterte GUI-Funktionalität für User Tasks

Im bestehenden System wird aus den Activiti spezifischen Form Properties ein entsprechendes html-Formular als Oberfläche generiert. Dies hat vor allem folgende Schwachpunkte:

- Müssen viele Daten in einem Task erfasst werden, die getrennt erfasst werden können, oder sind im Zuge einer Bearbeitung Auskunftsmöglichkeiten erforderlich, ist eine einzige Formularoberfläche nicht ausreichend und es müssen dafür weitere Tasks modelliert werden. Diese bilden eine Differenz zwischen dem Workflow-Modell und dem fachlichen motivierten Geschäftsprozessmodell, da sie aus technischer Limitierung resultieren. Dies kann die Durchlaufzeit eines Prozesses, je nach Häufigkeit solcher Fälle, stark beeinträchtigen.
- Die Ausgabe von Prozessvariablen erfolgt auf Basis der überschriebenen toString()-Methode aus der Klasse Object in Java. In den Form Properties ist für Variablen nur ein String-Feld für deren Wert vorhanden, welches mit dem Resultat der Methode gefüllt wird. Die toString()-Methode kann zwar mit einer statischen Hilfsvariable überladen werden, um verschiedene Ausgaben zu erzeugen, wie z.B. direkte Ausgabe von HTML-Code für eine Tabellenvariante in einer eigenen Listenimplementierung, aber durch Manipulation einer global gültigen Klassenvariable und möglicherweise viele verschiedene Zwecke ist in diesem Konzept keine ausreichende Lösung zu finden. Die Oberflächen können also nicht auf Basis von Form Properties generiert werden, will man z.B. entsprechende Tabellen oder Variablendetails anzeigen.

Um diese Probleme zu lösen, werden die Generierung der Oberflächen sowie die Verwendung von Form Properties verworfen. An Stelle dessen erhält jeder User Task seine eigene Bean für die Erzeugung von Oberflächen. Diese TaskUI-Beans sollen vom Zuordnungsprinzip genau wie die TaskWorker-Beans aus dem Konzept funktionieren und mit diesem interagieren. Diese soll auch auf Basis von Formularen funktionieren, allerdings können beliebig viele Formulare auf einer Oberfläche, sowie mehrere Oberflächen pro Task, erzeugt werden. Zu diesem Zweck werden im TaskWorker-Bean Aktionen definiert, welche im Zuge des Tasks ausgeführt werden können.

Beispiel: Mögliche Aktionen in der Belegerfassung wären Belegauskunft einholen, welches eine Maske mit allen Belegen zum Vertrag aufruft, Beleg erfassen, welches die

eingegebenen Daten validiert und Erfassung abschließen, welches eine Gesamtvalidierung durchführt und den Task abschließt.

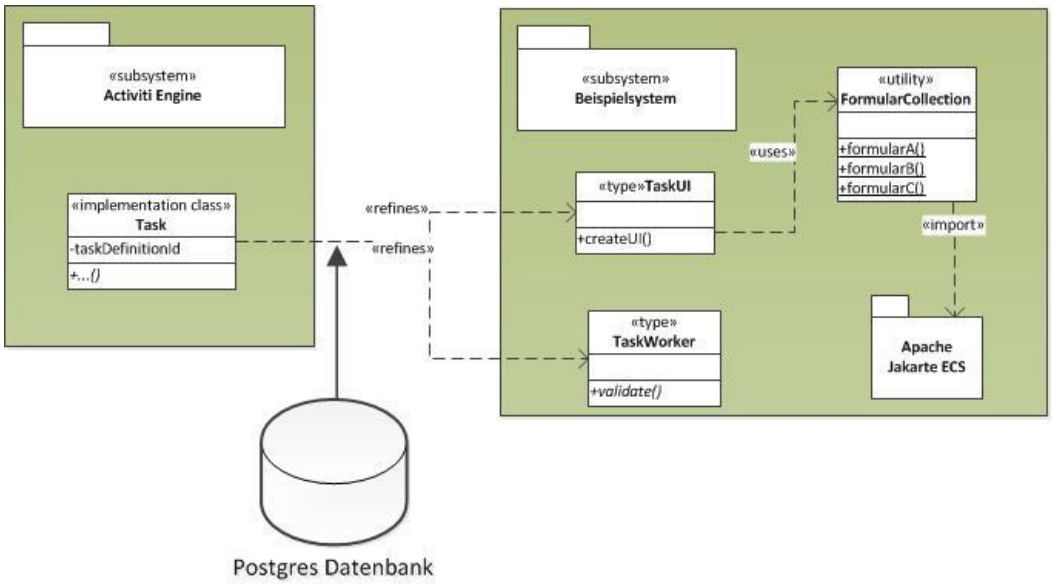


Abbildung 12 User-Task-Modell nach Erweiterung
Taskmodell nach erneuter Erweiterung. Abbildung erfolgt über das Attribut „taskDefinitionId“ in Typ Task. Das Mapping liegt dabei in zwei Datenbanktabellen.

Die TaskWorker-Bean gibt nach erfolgreicher Validierung die ausgeführte Aktion zurück, im Fehlerfall wird eine Exception geworfen.
In der TaskUI-Bean werden statisch alle erzeugbaren Masken in einem Enum definiert und die entsprechende Zuordnung von ausgeführter Aktion und Maske in einer Map gespeichert. Es wird dabei sowohl die aktuelle Maske als auch die zuletzt ausgeführte Aktion im Objekt in einem Cache gehalten. Die Methode zur Erzeugung von Oberflächen ist mit der durchgeführten Aktion und einer möglicherweise bestehenden Fehlernachricht parametrisiert. Im Normalfall wird die mit der Aktion verbundene Maske erzeugt und zurückgegeben, ansonsten wird die vorherige Maske mit einer Fehlermeldung angezeigt. Korrekt erfasste Daten aus den Formularen werden in Prozessvariablen persistent gehalten.

Umsetzung mit Fallbeispiel:

Die Umsetzung des Konzepts soll auf Basis eines Beispielprozesses erfolgen, welcher aus einem einzigen User Task besteht. Dieser Task soll in zwei Masken umgesetzt werden. In Maske A können in zwei Formularen Daten erfasst und gelöscht werden. Diese Daten bestehen aus je einem numerischen Wert und einem Tex.t Bereits erfasste Datensätze werden in einer Tabelle angezeigt. Maske B besteht aus einer Beispielausgabe in Form eines Textes. Zwischen den Masken kann per Aktion navigiert werden. Die Aufgabe kann nur dann abgeschlossen werden, wenn mindestens ein Datensatz je Formular in Maske A erfasst wurde.

6.5 Fehlerbehandlung und Robustheit

Tritt bei der Ausführung eines Prozesses ein technischer Fehler auf, so muss der Prozess in den letzten korrekten Zustand zurückgesetzt werden. Ein technischer Fehler ist dabei bedeutungsgleich mit einer ausgelösten Java Exception. Diese Exceptions können verschiedene Ursachen und Formen haben.
In Activiti ist die Prozessausführung in Jobs (siehe 6.3.3) eingeteilt. Tritt ein technischer Fehler auf, so wird der Prozess in den Zustand vor Ausführung der letzten Aktivität im vorherigen Job zurückgesetzt. Wurden allerdings Daten im fehlerhaften Job geschrieben, die sich außerhalb des Prozesskontexts befinden, so werden diese Daten nicht gelöscht. Es empfiehlt sich daher generell Daten in Prozessvariablen zu speichern. Im BPMN Standard ist mit Compensation und Cancel Events ein Mechanismus definiert, der auch Datenbanktransaktionen außerhalb des Prozesskontexts im Fehlerfall ermöglicht[27]. Dieser Mechanismus wird leider bisher nur experimentell von Activiti unterstützt[28] und funktioniert noch nicht. Die Verwendung von Prozessvariablen kann aber ebenfalls für Probleme sorgen, wenn Prozessvariablen verwendet werden, die zu einem Zeitpunkt vor dem wiederhergestellten Prozesszustand liegen. Ob der wiederhergestellte Zustand einen korrekten Zustand darstellt, ist also von der Verwendung der Prozessvariablen und dem schreiben weiterer Daten abhängig. Die Robustheit der Prozessausführung ist also hauptsächlich von diesen Variablen oder Daten abhängig.

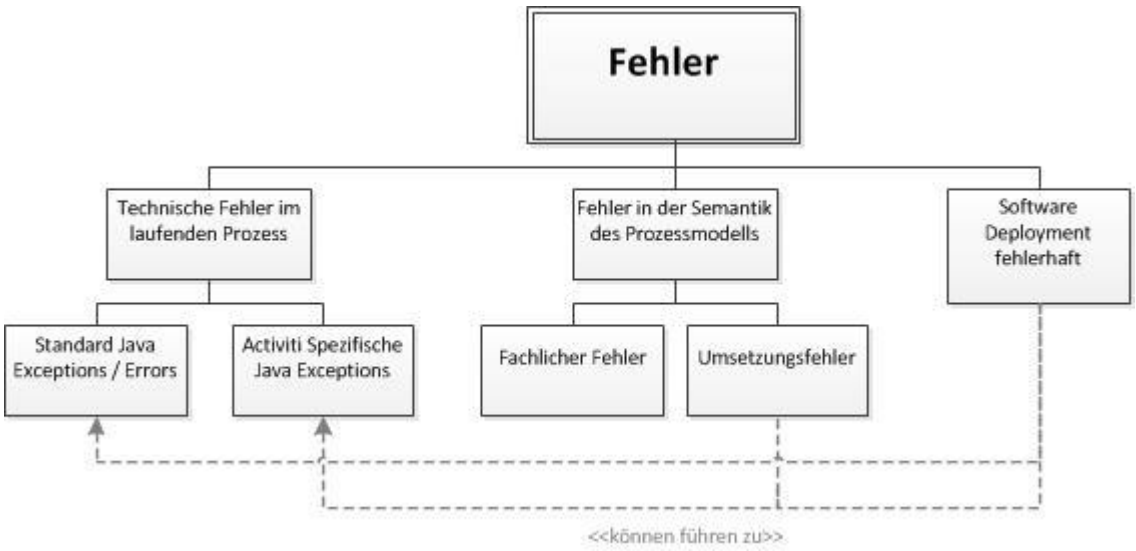


Abbildung 13 Fehlerkategorien

6.5.1 Fehlerursachen und -behandlung

Treten technische Fehler auf, so sind diese auf fehlerhafte Softwareartefakte oder Fehler im Prozessmodell zurückzuführen (siehe auch Abbildung 14).

[27] Es wird dazu im Fehlerfall ein Cancel Event ausgelöst. Durch das Auslösen dieses Events werden für sämtliche, bereits ausgeführte Tasks im (Sub-)Prozess Compensation Boundary Events ausgelöst, wenn solche vorhanden sind. An diese Events können dann Maßnahmen zur Kompensation durchgeführt werden. Genauere Beschreibung siehe [1, S. 153-158]
[28] Siehe [13]

- Fehler im Prozessmodell sind entweder fachlich falsche Modelle Vorgaben oder Fehler in der Umsetzung des Prozessmodells, wie z.B. falsche Verweise auf Klassen oder Prozessvariablen oder auch fehlende Tasks. Falsche Umsetzungen führen dabei nicht zu Fehlern, sondern bedeuten, dass ein fachlich falscher Prozess ausgeführt wird (Umgang mit fachlich falschen Prozessen siehe 6.5.2).
- Fehlerhafte Software bedeutet Fehler im Java-Code eines mit dem Prozess verbundenen Softwareartefakts, wie z.B. einer Service Task-Methode oder auch falsches Lesen bzw. Schreiben von Prozessvariablen.
- Fehler, die zu fehlerhaften Prozessinstanzen führen, erfordern stets eine Auslieferung (Deployment) von Software und/oder Prozessmodellen, um das Entstehen weiterer Fehler zu verhindern.

Wie im vorigen Abschnitt bereits erwähnt, können Fehler in Bezug auf Prozessvariablen zu fehlerhaften Zuständen bei der Zurücksetzung führen. Die gleiche Aussage kann für Prozessstrukturfehler getroffen werden, sollte z.B. im Modell ein Task fehlen oder eine falsche Verbindung durch Sequence Flow vorliegen. Fehler die nicht mit Prozessvariablen- oder struktur in Verbindung stehen lassen sich mit (Teil-)Auslieferung neuer Software oder eines neuen Prozessmodells beheben. Ein Beispiel hierfür ist die falsche Referenz einer Java-Klasse im Modell oder das Fehlen eines solchen Softwareartefakts. Prozessinstanzen, die durch einen solchen Fehler nicht weiter ausgeführt werden können, können nach einer Auslieferung weiterlaufen. Dasselbe gilt auch für fehlende Task Worker-Beans (5.18.3).
Ist ein fehlerhafter Prozesszustand entstanden, so lässt sich dieser nur bedingt beheben. Tritt der Fehler nach dem zurückgesetzten Zustand auf, so lässt sich dieser meist durch neue Auslieferung beheben, wobei diese eine Migration der fehlerhaften, laufenden Prozessinstanzen erfordert, sollte die Korrekturmaßnahme die Auslieferung eines neuen Prozessmodells beinhalten (siehe 6.5.2). Ist die Fehlerquelle vor dem zurückgesetzten Zustand gelegen, so sind aufwändigere und meist manuelle Maßnahmen, wie z.B. einen Datenpatch, notwendig, um laufende Prozessinstanzen weiterführen zu können. In einigen Fällen können die fehlerhaften Instanzen allerdings nicht korrigiert werden, wodurch diese gelöscht und neu angefangen werden müssen. Verschiedene Beispielfälle werden in Tabelle 10 aufgeführt, wobei in allen Fällen davon ausgegangen wird, dass die Fehlerquelle das Schreiben einer Variablen in einem Service Task ist.

Tabelle 9 Fehler im Prozess: Situationen, Quellen & Lösungen

Fehlersituation	Quelle*	Behebung
Eine referenzierte Variable wurde unter falschem Namen gesetzt. (NullPointer)	davor	Datenpatch oder Batchlauf, welcher die Fälle identifizieren kann und entsprechend den Namen der Variable ändert.
	danach	Korrektur in Software und neue Auslieferung.
Eine Variable wurde nicht gesetzt. Service Task ist aber vorhanden. (NullPointer)	davor	Keine Behebung möglich, da Wert verloren gegangen ist.
	danach	Korrektur in Software und neue Auslieferung.
Eine Variable wurde nicht gesetzt, da Service Task wegen Modellfehler nicht vorhanden ist. (NullPointer)	davor	Keine Behebung möglich. Wert war nie vorhanden. Zustand kann nicht wiederhergestellt werden in der aktuellen Prozessinstanz.
	danach	Prozessmigration notwendig, um Fehler in aktueller Instanz zu beheben.

6.5.2 Prozessmigrationen

Die Prozessmigration beschreibt den Vorgang, eine vorhandene Prozessinstanz in eine neue Prozessdefinition zu überführen. Eine solche Migration ist dann erforderlich, wenn ein Fehler, wie in den vorigen Abschnitten beschrieben, vorliegt und ein falscher Prozess durch einen neuen ersetzt werden muss. Dabei ist die Migration nur dann eine Lösung, wenn der Prozesszustand im Verlauf vor der Prozessanpassung liegt, da die einzige Migrationsmaßnahme die Änderung der Prozessdefinitionsversion auf Datenbankebene der Prozessinstanz ist. Diese Maßnahme ist auch nur dann erfolgreich, wenn die Process Engine zu diesem Zeitpunkt nicht läuft und der Prozess dementsprechend in einem stabilen Zustand ist. Nach erneutem Start der Engine wird die Prozessdefinition gelesen und entsprechend der Prozessverlauf bestimmt.
Eine Migrationsmaßnahme, die nach der entsprechenden Anpassung erfolgt, hat keinen Effekt und kann unter Umständen fehlerhaft sein, wenn ein solcher Zustand in der neuen Version nicht mit dem gleichen Verlauf erreichbar ist. Enthält der Prozesszustand Aktivitäten, welche nach der Korrektur nicht mehr vorhanden sind, ist eine Fortsetzung stets fehlerhaft.

6.6 Performance

Um die Performance der Engine, unabhängig von Geschäftslogik, Application Server und sonstigen Einflussgrößen, betrachten zu können, werden mehrere Prozessmodelle verwendet und in einem JUnit-Test vielfach zur Simulation verschiedener Systemlasten gestartet und die Durchlaufzeiten ermittelt. Dabei wird eine normale Activiti Engine, ohne CDI und Spring, in Verbindung mit einer In-Memory Database h2 (Version 1.2.132) verwendet. Da im bisherigen Verlauf stets die volle Historisierung verwendet wurde, wird diese Konfiguration ebenfalls auf den Test angewendet.

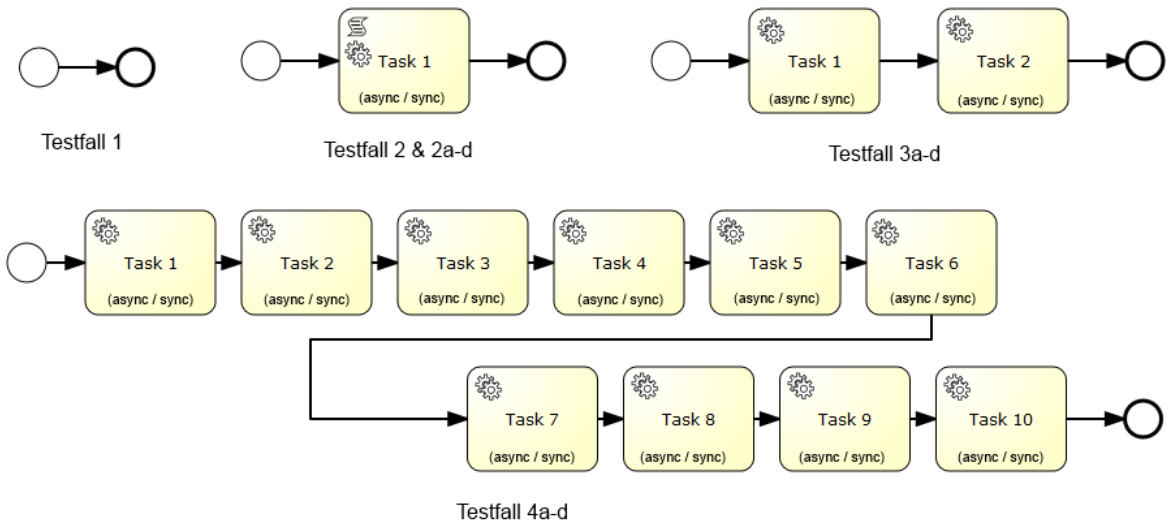


Abbildung 14Prozessmodelle der verschiedenen Fälle für den Performance Test

Testfall 1 und 2 sollen grundlegend ermitteln, wie lange die Activiti Engine benötigt, um einfache Aufgaben zu bewältigen. Dabei soll zuerst geprüft werden, wie viel Zeit das

Erstellen und Beenden eines Prozesses in Anspruch nimmt. Anschließend wird mit Testfall 2 geprüft wie viel Verzögerung eine Aktivität bewirkt. Dabei wird als erstes ein Script Task verwendet, da dieser komplett von der Engine ausgeführt wird und automatisch ist, so dass der gesamte Vorgang weiterhin in einer einzelnen Transaktion abläuft. Darauf folgend wird im Vergleich ein Service Task verwendet, welcher zunächst in den Fällen a und b eine leere Methode, anschließend eine Methode mit entsprechender Logik aufruft. Die Methode für den Dummy-Service-Task wird dabei über seine Klasse (implementiert JavaDelegate), der Logik-Service-Task über eine Spring Bean eingebunden. Dabei werden im Logik-Service-Task per Zufall 20 Strings mit einer Länge von 50 Zeichen generiert und in Key-Value-Paaren in einer Variablen gespeichert. Diese wird im zweiten Task ausgelesen und es wird erneut generiert, das Resultat an den Wert angehängt und die Variable erneut gespeichert. Dabei werden beide Service Tasks und die mit ihnen verbundenen Methoden sowohl in synchroner, als auch asynchroner Ausführung getestet.
Testfall 3 und 4 bestehen aus Test-Prozessen, die die Testkonstellationen aus den Testfällen 2a-d mit mehrfacher Ausführung der Service Tasks fortsetzen. Es wird davon ausgegangen, dass die asynchrone Ausführung im Vergleich schlechter abschneidet, da die Performance hauptsächlich von der erhöhten Anzahl von Threads und Transaktionen abhängen sollte. Ermittelt werden sollen die minimalen und maximalen Durchlaufzeiten der Prozessinstanzen, sowie die durchschnittliche Durchlaufzeit, wobei für die Messungen separate Start-, End- und Durchlaufzeiten im Test-Code ermittelt werden. Über die Aufteilung auf die Anzahl der ausgeführten Instanzen lässt sich somit der Durchschnitt in Millisekunden bestimmen. Auf Testmodelle mit mehreren Pfaden und Subprozessen wird verzichtet, da diese ebenfalls auf Ausführung mehrerer Jobs basieren. Da der JobExecutor, welcher für die Ausführung zuständig ist, diese letztendlich sequentiell abarbeitet[29], sollten die Auswirkungen auf die Performance durch die Ausführung bereits durch die Testfälle mit asynchroner Ausführung abgedeckt sein.

Aus der Durchführung der Tests ergaben sich folgende Werte für Durchlaufzeiten und Anzahl benötigter Ausführungen, um stabile und aussagekräftige Werte zu erhalten:

Tabelle 10 Performancemessungen

Fall	Beschreibung	Dauer (⌀ / ∑)	Max.	Min.	Fälle
1	Prozess aus Start und End Event	3.678 /33101 (4.53 / 40774)	10 (25)	3 (3)	9000
2	Prozess mit einem leeren Script Task	5.408 / 48672 (6.366 / 57294)	72 (40)	4 (5)	9000
2a	Prozess mit einem Dummy-Service Task	4.287 / 38580 (5.373 / 48353)	9 (28)	4 (4)	9000
2b	Prozess mit einem Dummy-Service Task (asynchron)	6.084 / 54754 (6.88 / 61920)	734 (42)	5 (5)	9000
2c	Prozess mit einem Logik-Service Task (asynchron)	6.776 / 60988 (7.737 / 69640)	13 (52)	6 (6)	9000
2d	Prozess mit einem Logik-Service Task	4.939 / 44453 (6.002 / 54021)	14 (57)	4 (5)	9000
3a	Prozess mit zwei Logik-Service Tasks	5.735 / 51622	12	5 (5)	9000

[29] Dies geht aus der Beschreibung des JobExecutors [10] und wurde durch die Beobachtungen in Abschnitt 6.3.3, sowie auf Rückfrage an das Activiti Entwicklungsteam () bestätigt.

		(6.779 / 61008)	(37)		
3b	Prozess mit zwei Logik-Service Tasks (asynchron)	9.799 / 88191 (10.543 / 94887)	571 (63)	9 (9)	9000
3c	Prozess mit zwei Dummy-Service Tasks (asynchron)	8.136 / 73224 (9.074 / 81664)	17 (50)	7 (7)	9000
3d	Prozess mit zwei Dummy -Service Tasks	5.175 / 46571 (5.917 / 53250)	623 (29)	4 (5)	9000
4a	Prozess mit zehn Logik-Service Tasks	11.771 / 101883 (12.671 / 114039)	469 (228)	10 (11)	9000
4b	Prozess mit zehn Logik-Service Tasks (asynchron)	31.394 / 282547 (32.981 / 296832)	327 (334)	15 (30)	9000
4c	Prozess mit zehn Dummy-Service Tasks (asynchron)	25.046 / 225415 (25.393 / 228544)	276 (163)	23 (23)	9000
4d	Prozess mit zehn Dummy -Service Tasks	10.755 / 96792 (10.892 / 98027)	37 (38)	9 (9)	9000

Ergebnis-Tabelle: Die Werte wurden in Millisekunden gemessen. Die Testergebnisse wurden mit einem durchschnittlichen PC erzielt, mit einem 4-Kern Prozessor und 8 GB Arbeitsspeicher.

Die Testausführung erfolgte dabei durch jeweils eine JUnit-Testmethode pro Fall, wobei die Resultate in Klammern von einzelnen Ausführungen der Methoden stammen und die anderen Resultate bei einer Ausführung am Stück angefangen mit 4d entstanden sind. Die maximalen Durchlaufzeiten fanden sich immer unter den ersten ca. 1-5 Ausführungen, die minimalen Werte ergaben sich erst im Bereich von 2500 – 3500 Ausführungen. Es wurden 9000 Ausführungen deshalb gewählt, um die statistischen Durchschnittswerte an einen hohen Betrieb und damit eine höhere Gewichtung auf die Minimalwerte zu legen. Der Unterschied zwischen minimalen und maximalen Durchlaufzeiten ist vermutlich in der Java Virtual Machine (JVM) begründet. Diese benötigt zu Anfang einer neuen Test-Methode bzw. einer neuen JVM-Instanz höhere Durchlaufzeiten, wobei eine neue Instanziierung dieses Phänomen stärker zeigt als der Methodenwechsel. Diese Vermutung wird durch die Messwerte bestätigt, die zeigen, dass Fall 4d, welcher direkt nach Start der JVM ausgeführt wurde, in beiden Verläufen fast gleiche Durchschnittszeiten zeigt, während alle anderen Fälle in der Einzelausführung höhere Durchlaufzeiten haben.
Die stark abweichenden und mit rot markierten Ergebnisse ließen sich nicht in weiteren Verläufen reproduzieren und werden daher als nicht relevant für die Untersuchung betrachtet.
Aus den Resultaten lassen sich folgende Annahmen ableiten:

1) Die Ausführung verschiedener Aktivitäten, wie Service Tasks oder Script Tasks, benötigt unabhängig vom Inhalt unterschiedlich lange. Ein Service Task ist im Schnitt ~1 ms schneller, als ein Script Task.
2) Aus dem Vergleich zwischen den Durchschnittswerten aus den Fällen 1, 2a, 3d und 4d ergibt sich:
   1 Service Task benötigt zwischen 4.287–3.678=0,609 und 5,373-4.53=0,843 ms
   2 Service Tasks benötigen zwischen 5.175-3.678=1,497 und 5.917-4.53=1,387 ms
   10 Service Tasks benötigen zwischen 10,755-3.678=7,077 und 10,892-4.53=6,362 ms
   Bildet man darüber die Summe der Durchschnittswerte für einen Service Task mit jeder Konstellation, ergibt sich, dass ein Service Task ~0,707 ms zur Ausführung

benötigt. Die Durchlaufzeiten mehrerer Service Tasks in einer Transaktion sind damit, unter Abweichung von ca. +/- 0,1ms konstant.

3) Vergleicht man die Resultate der asynchronen und der synchronen Ausführung der Prozesse mit Dummy-Service-Task-Elementen, so zeigt sich, dass die asynchronen Ausführungen im Schnitt pro Task 1,54 ms länger benötigen. Diese Zeit wird als Durchschnittswert festgelegt, die eine separate Job-Ausführung benötigt, wobei der Wert bei einer Abweichung von ca. +0,2/-0,1 ms konstant ist.
   Differenz der Durchschnittswerte (gerundet): 2b, 2a=1,797 (1,507) ms; 3d, 3c=1,48 (1,576) ms; 4d, 4c=1,429 (1,45) ms
4) Die Ausführung von Logik scheint das Ergebnis, unabhängig von der Anzahl der auszuführenden Tasks im Bereich von ca. +0,7-1 ms zu verlängern. Ruft man die Methoden allerdings, wie bei den Dummy-Service-Tasks, über die Klasse auf, zeigt sich, dass dieser konstante Overhead bei der Ausführung eines Tasks komplett und bei der Ausführung von 10 Tasks fast verschwindet. Eine entsprechende Untersuchung der Durchlaufzeit der Methode, ohne Variablen zu schreiben oder zu lesen, ergibt eine Durchlaufzeit von ca. 0.03 ms pro Methode. Der Overhead entstand also durch die Verwendung von Spring Beans. Die Task-Durchlaufzeit bleibt dabei unabhängig von der Menge der Variablenzugriffe so gut wie konstant[30]. Diese werden in der Job-Transaktion mit der Datenbank mit geschrieben und erzeugen nur einen minimalen Overhead.

Die Ableitungen führen zu dem Schluss, dass die Activiti Engine einen kaum für den Benutzer spürbaren Overhead erzeugt, welcher von der Anzahl der Datenbanktransaktionen abhängt. Im Vergleich zu der einfachen Methodenausführung zeigt sich zwar ein deutlicher Unterschied, welcher allerdings durch die Schaffung des Prozesskontexts mit festen Zuständen, Variablenspeicher und Historisierung gerechtfertigt ist. Entscheidend für die Anzahl der Transaktionen sind dabei der Grad der Historisierung und die Anzahl, sowie Ausprägung, der Jobs, aus denen einen Prozess besteht, wobei ebenfalls messbare Einflüsse durch Spring und dem Persistieren des eigentlichen Prozessobjekts entstehen. Diese Ergebnisse decken sich mit den Resultaten von Joram Barrez (siehe [5]), welche einen starken Performancegewinn der Engine zeigen, wenn eine Oracle Datenbank verwendet wird.

[30] Ausnahme ist hierbei das Lesen von Variablen nachdem die Engine neugestartet wurde. Dies erfordert eine geringe Verzögerung zum Lesen der Datenbankspalte und zum Deserialisierung des Objekts.

7 BPMN Elemente als Teile einer Anwendung

7.1 User Tasks

Der User Task stellt als Element in Geschäftsanwendungen einen manuell durchgeführten Schritt im Prozess dar, welcher durch UserEingaben und eventuell sich anschließender automatischer Plausibilitätsprüfung erfüllt wird. Erzeugt wird der Task durch die Process Engine. Die eigentliche manuelle Tätigkeit liegt beim Menschen und der Beschaffung der Daten. Es ist dabei nicht festgelegt, ob ein solcher Task aus mehreren Eingabemasken und Formularen besteht oder aus nur einer einzigen Maske mit einem Formular. Als Beispiel kann dafür die Belegerfassung in der Leistungsabrechnung verschieden gestaltet werden. Dieser Subprozess ist fein granular strukturiert. Das bedeutet, die eigentliche Aufgabe wurde in einer hohen Anzahl von Tasks modelliert. Im bestehenden System führt die Erfassung mehrerer Belege und Belegpositionen zu einem hohen Aufwand durch das wiederholte Aufnehmen und Abschließen von Tasks in Verbindung mit der Overview-Komponente. Eine solch starke Unterteilung ist nur dann sinnvoll, wenn die einzelnen Teilaufgaben von unterschiedlichen Personen ausgeführt werden. Das trifft im Fall der Belegerfassung nicht zu.

Abbildung 15 stellt eine alternative Beleg-erfassung dar. Sie besteht aus 3 User Tasks und einem Service Task. Die Belegerfassung umfasst in diesem Fall die Erfassung, Bearbeitung und Löschung von sowohl Belegen als auch dazugehörigen Positionen. Umgesetzt werden kann dieser Task z.B. durch eine Maske mit mehreren Formularen. Eine Tabelle zeigt alle Belege an. Pro Beleg stehen Buttons zur Bearbeitung, Löschung und Auswahl zur Verfügung. Ein Formular ermöglicht die Erfassung neuer und die Bearbeitung vorhandener Belege in Verbindung mit dem Button. Die Auswahl eines Beleges führt dazu, dass eine weitere Tabelle mit den entsprechenden Positionen gefüllt wird. Dies

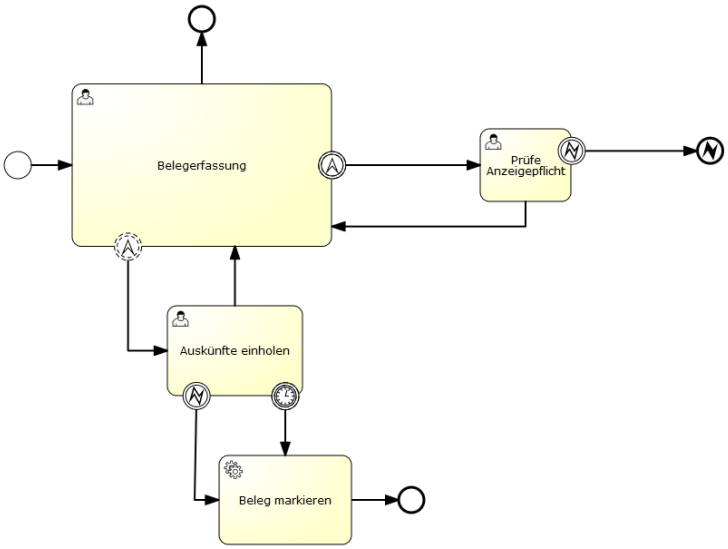


Abbildung 15 Alternativer Sub-Prozess für die Belegerfassung

kann bei Platzknappheit auch auf einer eigenen Maske umgesetzt werden. In dieser Tabelle stehen dann dieselben Optionen zur Verfügung wie für einen Beleg, abgesehen von der Auswahl. Stellt der Sachbearbeiter fest, dass Auskünfte benötigt werden oder Verdacht auf eine Anzeigepflichtverletzung besteht, kann er durch ein Formular angeben, für welchen Beleg dieser Bedarf besteht und durch einen Button ein Escalation Event auslösen. Beide Aktivitäten unterbrechen die Belegerfassung. Im Normalfall wird nach der jeweiligen Aktivität jedoch die Erfassung weitergeführt. Sollten jedoch im Verlauf dieser Tasks die entsprechenden Timer oder Error Boundary Events ausgelöst werden, wird entweder die Erfassung und im Verlauf des Error Event Handlings bei Anzeigepflichtverletzung der gesamte Prozess abgebrochen. Die entsprechenden Error Events werden erneut durch Oberflächenelemente ausgelöst. Ein Service Task markiert Belege im Fall der nicht gewährten Auskunft oder als Resultat der Auskunft als nicht abrechenbar.
Der neue Subprozess hat eine deutlich geringere Anzahl von Tasks. Dies führt zu einer verstärkten Verlagerung in eine Controller Komponente (im MVC Sinne) für die entsprechende Oberfläche. Eine stärkere Aufteilung ist nicht möglich, da die Anzeigepflicht im erweiterten Sinne nur von einem Teamleiter festgestellt werden kann und eine echte Zeitbegrenzung für die Auskünfte eingehalten werden muss. Ein Vergleich beider Varianten soll nun unter der Annahme gemacht werden, dass die Ursprungsvariante in einem kontinuierlichen Verlauf ausgeführt wird. Eine beispielhafte Gestaltung ist im Umsetzungskapitel (6.4.1) beschrieben.

Tabelle 11 Aufgabenstrukturierungsaspekte

Aspekt	Variante Fallbeispiel	Neue Variante
Historische Daten und Analyse	Die Analyse ist auf Basis von historischen Prozess-, Task und Event Daten möglich. Sie wird automatisch durch ein Framework wie Activiti ermöglicht.	Muss eigens implementiert werden und in Verbindung mit Prozessdaten analysiert werden.
Verständlichkeit und Bezug auf das System	Klar verständlicher Verlauf. Kann durch semantisch irrelevante Elemente weiter gesteigert werden. Die Struktur kann direkt als Grundlage für Diskussionen und Verbesserungsvorschläge genutzt werden.	Gesamtstruktur im größeren Sinne verständlich, allerdings muss die Oberfläche selbst analysiert und mitbewertet werden. Zusätzliche Daten für Diskussion erforderlich.
Durchsatz / Benutzbarkeit	Masken sind jeweils klar verständlich, allerdings führt die feine Strukturierung in vielen Tasks dazu, dass zu einem bestimmten Zeitpunkt nur Daten für einen Beleg oder eine Position erfasst/korrigiert werden können. Diese Aufgabe muss stets vor weiteren Schritten abgeschlossen werden. Dieser Mangel an Flexibilität kann in einer höheren Durchlaufzeit resultieren.	Die Strukturierung der eigentlichen Benutzertätigkeit kann in einem eigenen Konzept vorgenommen und getestet werden. Dadurch kann eine gegebenenfalls bessere Benutzbarkeit und damit auch ein höherer Durchsatz erzielt werden.

Aus der Tabelle geht hervor, dass die unterschiedlichen Varianten in ihren Extremen Vor- und Nachteile haben können, was aber von den jeweiligen Anforderungen abhängt. Will man auf Basis von Prozessdaten beispielsweise ermitteln, wie hoch die Fehlerrate der Benutzer bei der Erfassung ist, wie die durchschnittliche Verarbeitungszeit für einen Task ist oder ob die Aufteilung und Abfolge sinnvoll gestaltet ist, so ist eine feine Granularität hilfreich und erforderlich. Eine niedrige Granularität kann jedoch die Benutzerfreundlichkeit eines Systems fördern, verlagert dafür aber mehr Logik und separaten Konzeptionsaufwand in eine einzige Aufgabe, so dass das BPMN Modell weniger aussagekräftig wird. Es hängt also davon ab, welche Kriterien bei der Entwicklung und Verwendung besonders wichtig sind.

Aus den Betrachtungen zeigt sich, dass das User Task Element in seiner Definition nur eindeutig definiert ist, als der Zeitraum, zwischen seiner automatischen Erstellung und seiner Beendigung durch den Bearbeiter. Die Eindeutigkeit und Schwierigkeit seiner genauen Definition ist abhängig von der Wahl seiner Größe und der Aufgabe selbst. Für die Belegerfassung ist es effektiver eine niedrigere Granularität zu wählen. Es handelt sich bei manuellen Belegerfassungen um einfache Tätigkeitsabfolgen, bei der ein hoher Durchsatz wichtig ist, da in einer Versicherung in Normalfall viele Rechnungen eingehen. Ein IT-System soll diese Tätigkeit effektiver machen und weniger Personal erfordern, um die Kosten zu minimieren. Eine genaue Untersuchung des Ablaufs ist in diesem Fall nicht interessant, da der Ideale Ablauf eindeutig und einfach zu definieren ist.

7.2 Service Tasks

Service Tasks sind automatische Elemente, die ohne Warten auf Benutzeraktionen durchgeführt werden. In Activiti sind diese Elemente als Methodenaufruf interpretiert. Nach BPMN-Spezifikation ist diese Interpretation valide, wobei sie im breiteren Sinne die Bearbeitung einer Aufgabe durch eine Maschine beschreibt. Ob dieser Methodenaufruf durch Verweis auf eine implementierende Klasse des Interfaces Java Delegate oder durch eine Activiti Expression realisiert ist ist eine Stilfrage und hängt von strukturellen Präferenzen, unter Umständen auch von technischen Frameworks ab. Im Fallbeispiel wurde die Entscheidung gefällt Expressions zu verweden, um Methoden in Klassen zu gruppieren, welche wiederrum eine Klammer um einen Subprozess und dessen weitere Hierarchieebenen bilden. Die Tasks werden fachlich in Klassen gruppiert, wie es im Fallbeispiel anhand der Subprozesse Erstattungsprüfung und Erstattungsberechnung vorgenommen wurde.
Methodenaufrufe selbst können sich sehr in ihrer Komplexität unterscheiden, da nicht festgeschrieben ist, ob innerhalb der Methode weitere Methoden aufgerufen werden können und keine Beschränkung der Methodenlänge vorgegeben ist. Die Granularität hängt, wie zuvor für User Tasks beschrieben, von der Wahl der Aufgabengröße und der fachlichen sowie im Fall der Service Tasks auch von der technischen Komplexität ab.
Einfluss auf die Entscheidungskriterien bei der Aufgabengestaltung haben mehrere Faktoren. Zum einen ist es das bereits bekannte Kriterium der Analyse von historischen Prozessdaten. Diese können verwendet werden, um Performance bewerten zu können oder eine Analyse der Durchlaufpfade vorzunehmen, welche in einer Restrukturierung

resultieren kann. Zum Beispiel wird in der Erstattungsprüfung eine Reihe von Regelprüfungen vorgenommen, nach denen eine Belegposition in verschiedene Teile unterteilt wird, wenn ein entsprechender Tarif gefunden wurde. Die Prüfung erfolgt in sequentieller Reihenfolge. Es wird geprüft, ob weitere tarifliche oder vertragliche Faktoren den Leistungsanspruch mindern oder ausschließen. Dabei werden der Reihe nach

1) der Zeitraum der tariflichen Gültigkeit
2) tarifliche Wartezeiten
3) Leistungsausschlüsse

geprüft. Nach jeder Prüfung wird entschieden, ob weiterer Prüfungsbedarf besteht. Kein Anlass für weitere Prüfungen ist gegeben, wenn kein (Teil-)Anspruch auf Erstattung mehr besteht. Die sequentielle Anordnung erfolgt auf Basis von Logik. In der Praxis kann sich jedoch zeigen, dass statistisch ein Leistungsausschluss am häufigsten als Grund von Nicht-Erstattung auftritt. Jede Prüfung erfolgt auf Basis von Vertrags- und Tarifdaten, und muss daher auf die Datenbank, unter Umständen auch auf Fremdsysteme, zugreifen und entsprechende Logik ausführen. Bei entsprechenden Datenmengen können also statistisch irrelevante Prüfungen durch ihre Platzierung im Verlauf einen Performanceverlust bedeuten, der den Arbeitsfluss ausbremsen kann. An dieser Stelle kann durch die Auswertung von historischen Prozessdaten eine entsprechende Erkenntnis gewonnen und eine Umstellung im Prozess vorgenommen werden.
Die Aufgabendefinition kann aber auch von rechtlichen oder wirtschaftlichen Rahmenbedingungen abhängen. Damit ist z.B. eine zeitliche Beschränkung einer bestimmten Aktivität gemeint, wie Sie im Rahmen der Einholung von Auskünften auftreten kann. Diese muss beispielsweise innerhalb von 14 Tagen geschehen, sonst wird der Leistungsanspruch nicht weiter geprüft oder eine Eskalation vorgenommen. Es kann aber auch eine verschiedenartige Behandlung eines Events je nach Zeitpunkt des Auftretens im Prozess erforderlich sein, welche eine bestimmte Aufgabenteilung erforderlich macht. In diesem Fällen müssen für entsprechende Abschnitte mehrere Tasks definiert werden, obwohl aus technischer Sicht, z.B. durch eine gebündelte Datenbankanfrage an Stelle mehrerer Anfragen, eine gröbere Aufteilung sinnvoller wäre.
Die Verständlichkeit des Prozessmodells ist ebenfalls ein Faktor. Bei beiden Task-Typen sollte die Wichtigkeit dieses Faktors nicht unterschätzt werden, da ein Prozessmodell, im Gegensatz zu einem rein technischen Modell, vor allem auch Grundlage für die eigentliche Prozessgestaltung und daher Gegenstand, auf Basis dessen Diskussionen stattfinden und geschäftsrelevante Entscheidungen getroffen werden, ist.

7.3 Events

Events, wie sie in BPMN verwendet werden, stellen, wie Service und User Tasks, ein Konzept dar, welches außerhalb vom BPMN und der Activiti Engine in der Informatik seit längerer Zeit Anwendung findet. Die Prozess Engine ist dabei der Event-Bus und löst Events entsprechend aus, indem sie registrierte Listener, die einem Event im Kontext eines Prozesses zugeordnet sind, informiert.
Bei der Integration dieses Konzepts in Prozessmodelle werden den Events verschiedene Spezialisierungen hinzugefügt:

- Start und End Events im Prozess: Mehrere Events für Start und Ende möglich

- Publish-Subscribe-Pattern[31] (1 : 0..n) Events: Ein Auslöser, mehrere oder keine wartenden Abonnenten
- Point-to-Point (1 : 0..1) Events: Ein Auslöser, ein oder kein wartender Abonnent
- Einzel-Events: Ausgelöst, ohne weitere wartende Events
- Events alleinstehend im Prozessverlauf
- Events, die im Rahmen einer Aktivität auftreten können und bei Auslösung einen separaten Pfad beschreiten
    - Separater Pfad kann inner- oder außerhalb der Aktivität liegen.
    - Event kann die Aktivität abbrechen oder weiterlaufen lassen

Events können ihren Auslösetyp spezifizieren, wie das Auftreten eines spezifizierten Geschäftsfehlers oder das Eintreffen einer bestimmten Nachricht. Zwischen Spezialisierung und Auslösetyp existiert mit der BPMN-Spezifikation ein Regelwerk, welches festlegt, welche zulässigen Kombinationen existieren.

7.4 Bewertung

Die beschriebenen Elemente entsprechen Konzepten, welche in der Softwareentwicklung bereits Anwendung finden. User Tasks sind technisch gesehen GUI-Komponenten (Graphical User Interface) oder eine beliebige andere Art von Benutzerschnittstellen, Service Tasks repräsentieren den Aufruf eines Softwareartefakts bzw. eines Automaten und Events sind als solche ebenfalls bekannt, aber mit einer spezifischen Semantik versehen. Aus technischer Sicht finden sich in der BPMN diese und weitere bekannte Konzepte wieder. Die Elemente werden nach den Vorgaben und Regeln der BPMN-Spezifikation bei der Prozessmodellierung eingesetzt um einen ausführbaren Prozess zu erstellen.
Die Menge an Elementen und Einsatzregeln werden allerdings für manche Bereiche bzw. Einsatzzwecke als unzureichend oder als unpräzise spezifiziert bewertet[32], wie z.B.:

- Beim Auftreten mehrerer eingebetteter Interrupting Events ist nicht definiert, ob alle Events ausgelöst werden und in welcher Reihenfolge sie ausgeführt werden (vgl. [6, S. 306-307])
- Die Modellierungsmöglichkeiten der Kommunikation bzw. Interaktion bei der konkurrierenden Ausführung von unabhängigen Subprozessen oder Prozessen unterschiedlicher Unternehmen oder unterschiedlicher Bereiche in einem Unternehmen sind unzureichend (vgl. [6, S. 307]).
- Obwohl z.B. Bruce Silver die ausführbare Ebene der BPMN (siehe Abschnitt 2.1.3) sowie die Erstellung beschreibenden bis hinzu ausführbaren Modellen beschreibt (siehe 2.1.4), liegt in der Spezifikation kein konkretes Vorgehen für die Verfeinerung von Modellen und damit keine Garantie vor, dass ein Modell tatsächlich korrekt ausführbar ist (vgl. [6, S. 307-308]).

[31] Definition vgl. [21, S. 115] „2. THE BASIC INTERACTION SCHEME", 1. Absatz
[32] Nach Egon Börger, siehe [6, S. 306ff] Abschnitt „2 Problems of BPMN 2.0"

8 Fazit

Wie an der Umsetzung (Abschnitt 6) zu sehen ist, werden nicht alle Elemente derzeit von der Activiti Engine unterstützt, sowie nicht alle unterstützten Elemente korrekt ausgeführt. Des Weiteren existieren anhand von Activiti-spezifischen Eigenschaften bei der Modellierung Abweichungen zum BPMN Standard. Diese Faktoren deuten darauf hin, dass die Auswahl des Frameworks, im speziellen der Process Engine, Einfluss auf die Modellierung von Prozessen nehmen kann. Basierend darauf führt die Auswahl von Activiti zu einer Toolabhängigkeit, die durch das Fehlen eines standardisierten Daten- und Objektmodells für BPMN noch vergrößert wird. Eine Engine auszuwählen ist also eine strategische Entscheidung für den IT-Sektor und damit für das gesamte Unternehmen. Es empfiehlt sich daher, eine sorgfältige Prüfung und eine Entscheidung, im Zweifel gegen ein Nischenprodukt, zu treffen, um Konflikte zu vermeiden. Sollte ein Wechsel der Process Engine erforderlich sein, sind die Kosten ansonsten sehr hoch, da Prozessanpassungen und aufwändige Migrationen notwendig sind. Weitere Faktoren, die bei der Auswahl des Frameworks beachtet werden sollten, sind etwaige integrierte Modellierungstools und die Möglichkeiten weitere Frameworks, z.B. zur Ausführung von Business Rules oder Process Monitoring. In Activiti können entsprechende, ausgewählte Frameworks (z.B. Drools für Business Rules) integriert werden, was zu einer weiteren Toolabhängigkeit führt. Der Activiti Designer kann nicht ersetzt werden und ist durch die Activiti-spezifischen Elemente teilweise inkompatibel zu anderen Modellierungstools, u.a. auch dem Activiti Modeler.

Entscheidet man sich für die Verwendung von BPMN-Modellen so ist dies aber keinesfalls eine reine Entscheidung für den IT-Sektor, sondern hat ebenso große Auswirkungen auf die gesamte Unternehmensstrategie. Voraussetzung für die Prozessplanung mit BPMN sollte eine komplette Integration in die involvierten Bereiche, von rein betriebswirtschaftlichen Planungsbereichen bis zur Detailplanung in der IT, vorausgehen (vgl. Workflow-Lifecycle [7, S. 57-61]). Nur so kann sichergestellt werden, dass die IT-gestützten Prozesse, die in vielen Unternehmen einen Großteil der operativen Tätigkeiten ausmachen, sich strikt an die betriebswirtschaftliche Planung halten. Eine solche, großflächige Integration der BPMN ist mit Erlernungsaufwand verbunden, der zu einer Anlaufzeit führt ist, bis die Methode effektiv funktioniert. Weitere Kosten können durch das Einbinden externer Fachleute entstehen.

Im Vergleich zwischen BPMN, eEPK (erweiterten Ereignisgesteuerten Prozessketten), Swimlane und weiteren Modellierungsmethoden wird BPMN eine hohe Komplexität und ein hoher Schlungsaufwand attestiert, wobei, im Vergleich zu eEPK, eine Standardisierung der Notation durch die OMG vorliegt. Die eEPK-Methode kann, durch ihren ebenfalls hohen Detailierungsgrad, als Hauptkonkurrent betrachtet werden. Ihr wird ein geringerer, jedoch trotzdem hoher, Schulungsaufwand zugeschrieben [7, S. 102]. Ein Vergleich der Nutzungshäufigkeit der genannten und weiteren Methoden aus dem Jahr 2007 zeigt allerdings einen deutlichen Vorsprung für die eEPK (43,1 % eEPK, 16,4 % BPMN; vgl. [7, S. 65]). Im Bereich der Workflow-Management-Systeme (Gesamtsysteme für die Verwendung

von Workflowmodellen zur Prozesssteuerung, vgl. Definition [7, S. 227-230]), zu denen man das Activiti Framework zählen kann, existieren mehrere Standardisierungsansätze. Anhand von [7, S. 231-233] zeigt sich allerdings, dass sich keiner der Ansätze komplett durchsetzen konnte, BPMN durch die Standardisierung des Formats in Version 2.0 allerdings einen Schritt in Richtung Standardisierung des Systems gemacht hat.

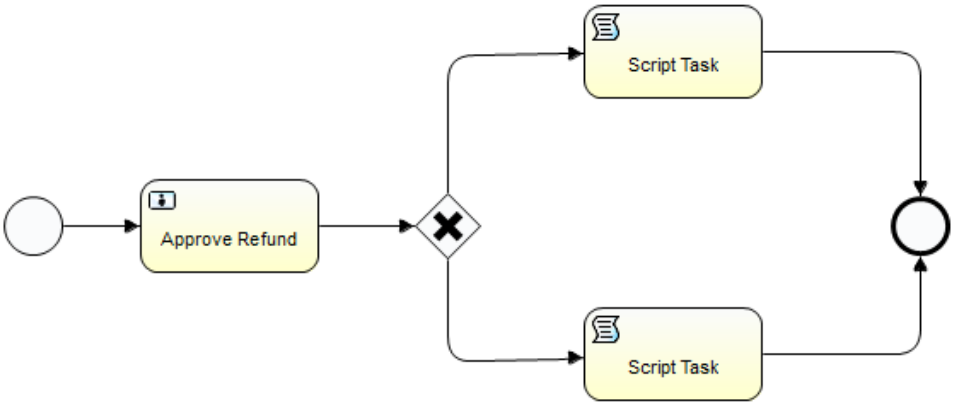
Anhand der im Rahmen dieser Arbeit entstandenen Beispielanwendung und der Verwendung von BPMN Modellen zeigten sich durchaus positive Eigenschaften einer Zentrierung auf Prozessmodelle. Diese liegen zum einen in der Notation selbst, dadurch dass diese im Besonderen von der Art der Elemente (vergleiche 6.1, 7) gut für eine Automatisierung durch Software geeignet ist, und durch die Strukturierungsmöglichkeiten durch Call Activities und Subprocesses, die eine einfache Aufteilung der Komplexität durch hierarchische Teilaufgaben ermöglichen. Zum anderen zeigte sich, dass viel Aufwand durch die Verwendung der Activiti Engine dadurch gespart werden konnte, dass die entstandene Beispielanwendung nur einen entsprechenden Rahmen für die Ausführung von Prozessen und User Tasks bieten musste. Der eigentliche Workflow wurde stets von der Engine durchgeführt. Betrachtet man also den Aufwand für die einmalige Integration der Engine mit der Implementierung eines eigenen Prozessmodells mit Ausführung und Historisierung, so lässt sich klar eine große Ersparnis an Aufwand erreichen. Die Implementierung der Tasks ist zwar, je nach Komplexität, aufwändig, allerdings ist die Aufgabe stets an richtiger Stelle durch das Modell angeordnet und kann leicht verschoben werden. Durch das grafische Modell und den Namen des Tasks, kann der Rahmen und Kontext einer Aufgabe stets gut nachvollzogen werden. Es muss allerdings erneut betont werden, dass eine schriftliche Spezifikation der Aufgabe dadurch nicht gänzlich ersetzt werden kann.

Zusammenfassend betrachtet ergibt dies ein gemischtes Urteil, da für das effektive Einsetzen von Modellen zur IT-gestützten Ausführung von Prozessmodellen entsprechende Aufwände durch Schulungen und Anpassungen in der Planung entstehen. Zusätzlich führt die Verwendung externer Software für die Ausführung dieser Modelle, die einen zentralen Aspekt in der Effektivität der operativen Tätigkeiten einnehmen, zu einem Risiko durch mögliche Toolabhängigkeiten. Sind diese Hürden gemeistert, lassen sich allerdings Aufwände in der Implementierung der Prozesse, vor allem der Steuerung, sparen und ein Mehrwert durch das Modell, welches der betriebswirtschaftlichen Planung entspricht, und Process Monitoring für die Prozessoptimierung erzielen.

8.1 Ausblick

Mit Blick auf die Zukunft ergibt sich durch die Standardisierung und die Erweiterungsmöglichkeiten der BPMN (z.B. durch die Einführung weiterer Spezialisierungen von Tasks oder Events; Beispiel Service Tasks und Business Rule Tasks, vgl. 6.1), eine gute Zukunftssicherheit, sollten Anpassungen der Notation notwendig sein. Würde sich ebenfalls ein Standard im Bereich der Workflow-Management-Systeme etablieren, so würden sich dadurch die Risiken einer Toolabhängigkeit stark reduzieren. Durch ein somit reduziertes Risiko bei der Verwendung von solchen Systemen ließe sich der Vorteil durch Unabhängigkeit und Flexibilität bei der Toolverwendung steigern.

Anhang A




```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:activiti="http://activiti.org/bpmn"
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
xmlns:omgdc="http://www.omg.org/spec/DD/20100524/DC"
xmlns:omgdi="http://www.omg.org/spec/DD/20100524/DI"
typeLanguage="http://www.w3.org/2001/XMLSchema"
expressionLanguage="http://www.w3.org/1999/XPath"
targetNamespace="http://www.activiti.org/test">
  <process id="checkManagementProcess" name="check Management Process" isExecutable="true"
activiti:candidateStarterUsers="fozzie">
    <startEvent id="startevent1" name="Start"></startEvent>
    <userTask id="approveRefund" name="Approve Refund" activiti:assignee="fozzie">
      <extensionElements>
        <activiti:formProperty id="contractId" name="Contract Identification"
variable="contractId" writable="false"></activiti:formProperty>
        <activiti:formProperty id="amountToBeRefunded" name="amount To Be Refunded"
variable="amountToBeRefunded" writable="false"></activiti:formProperty>
```

```
 <activiti:formProperty id="isSpecial" name="is Special  contract?" variable="isSpecial"
writable="false"></activiti:formProperty>
        <activiti:formProperty id="ci" name="contract informations" variable="ci"
writable="false"></activiti:formProperty>
        <activiti:formProperty id="isRelativeofEmployee" name="Is telative of Employee?"
variable="isRelativeofEmployee" writable="false"></activiti:formProperty>
        <activiti:formProperty id="approval" name="approval?" type="enum" required="true">
          <activiti:value id="false" name="oh hell no"></activiti:value>
          <activiti:value id="true" name="yepp-dideli-doo"></activiti:value>
        </activiti:formProperty>
        <activiti:formProperty id="explanation" name="Give explanation for decision!"
type="string" required="true"></activiti:formProperty>
      </extensionElements>
    </userTask>
    <endEvent id="endevent1" name="End"></endEvent>
    <sequenceFlow id="flow1" sourceRef="startevent1"
targetRef="approveRefund"></sequenceFlow>
    <scriptTask id="scripttask1" name="Script Task"></scriptTask>
    <scriptTask id="scripttask2" name="Script Task"></scriptTask>
    <exclusiveGateway id="exclusivegateway1" name="Exclusive Gateway"></exclusiveGateway>
    <sequenceFlow id="flow2" sourceRef="approveRefund"
targetRef="exclusivegateway1"></sequenceFlow>
    <sequenceFlow id="flow3" sourceRef="exclusivegateway1"
targetRef="scripttask1"></sequenceFlow>
    <sequenceFlow id="flow4" sourceRef="exclusivegateway1"
targetRef="scripttask2"></sequenceFlow>
    <sequenceFlow id="flow5" sourceRef="scripttask2" targetRef="endevent1"></sequenceFlow>
    <sequenceFlow id="flow6" sourceRef="scripttask1" targetRef="endevent1"></sequenceFlow>
  </process>
  <bpmndi:BPMNDiagram id="BPMNDiagram_checkManagementProcess">
    <bpmndi:BPMNPlane bpmnElement="checkManagementProcess"
id="BPMNPlane_checkManagementProcess">
      <bpmndi:BPMNShape bpmnElement="startevent1" id="BPMNShape_startevent1">
        <omgdc:Bounds height="35.0" width="35.0" x="250.0" y="270.0"></omgdc:Bounds>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape bpmnElement="approveRefund" id="BPMNShape_approveRefund">
        <omgdc:Bounds height="55.0" width="105.0" x="330.0" y="260.0"></omgdc:Bounds>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape bpmnElement="endevent1" id="BPMNShape_endevent1">
        <omgdc:Bounds height="35.0" width="35.0" x="770.0" y="270.0"></omgdc:Bounds>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape bpmnElement="scripttask1" id="BPMNShape_scripttask1">
        <omgdc:Bounds height="55.0" width="105.0" x="590.0" y="158.0"></omgdc:Bounds>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape bpmnElement="scripttask2" id="BPMNShape_scripttask2">
        <omgdc:Bounds height="55.0" width="105.0" x="590.0" y="335.0"></omgdc:Bounds>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape bpmnElement="exclusivegateway1" id="BPMNShape_exclusivegateway1">
        <omgdc:Bounds height="40.0" width="40.0" x="490.0" y="267.0"></omgdc:Bounds>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge bpmnElement="flow1" id="BPMNEdge_flow1">
        <omgdi:waypoint x="285.0" y="287.0"></omgdi:waypoint>
        <omgdi:waypoint x="330.0" y="287.0"></omgdi:waypoint>
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge bpmnElement="flow2" id="BPMNEdge_flow2">
        <omgdi:waypoint x="435.0" y="287.0"></omgdi:waypoint>
        <omgdi:waypoint x="490.0" y="287.0"></omgdi:waypoint>
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge bpmnElement="flow3" id="BPMNEdge_flow3">
        <omgdi:waypoint x="510.0" y="267.0"></omgdi:waypoint>
        <omgdi:waypoint x="510.0" y="187.0"></omgdi:waypoint>
        <omgdi:waypoint x="590.0" y="185.0"></omgdi:waypoint>
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge bpmnElement="flow4" id="BPMNEdge_flow4">
        <omgdi:waypoint x="510.0" y="307.0"></omgdi:waypoint>
        <omgdi:waypoint x="510.0" y="367.0"></omgdi:waypoint>
        <omgdi:waypoint x="590.0" y="362.0"></omgdi:waypoint>
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge bpmnElement="flow5" id="BPMNEdge_flow5">
        <omgdi:waypoint x="695.0" y="362.0"></omgdi:waypoint>
        <omgdi:waypoint x="787.0" y="362.0"></omgdi:waypoint>
```

```
<omgdi:waypoint x="787.0" y="305.0"></omgdi:waypoint>
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge bpmnElement="flow6" id="BPMNEdge_flow6">
        <omgdi:waypoint x="695.0" y="185.0"></omgdi:waypoint>
        <omgdi:waypoint x="787.0" y="185.0"></omgdi:waypoint>
        <omgdi:waypoint x="787.0" y="270.0"></omgdi:waypoint>
      </bpmndi:BPMNEdge>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

Anhang B

Auf der beigelegten CD finden sich, neben der Anwendung, die SQL Scripts für die Erstellung der benötigten Datenbanktabellen nach dem Datenmodell aus Abschnitt 5.13 und entsprechende Beispieldatensätze. Diese Datensätze beinhalten Tarifdefinitionen und Leistungsarten (siehe Abschnitt 3.4) sowie einen Beispielvertrag mit drei Versicherten Personen, die jeweils verschiedene Tarifkombinationen besitzen. Des Weiteren sind zwei Benutzer für die Engine und zwei verschiedene Benutzergruppen als Datensätze für die Activiti Datenbank (Version 5.12) und eine Beispielrechnung vorhanden.
In einer weiteren Datei finden sich

- Links zu den verwendeten Frameworks und Tools
- Die Testfälle für den Performance-Test

Literaturverzeichnis

[1] B. Silver, BPMN Method & Style, Aptos (CA, USA): Cody-Cassidy Press, 2009.

[2] E. Wolff, Spring 3. Framework für die Java-Entwicklung, Heidelberg: dpunkt.verlag, 2010.

[3] C. 1.-2. b. R. D. W3Schools, „HTTP Methods: GET vs. POST," Refsnes Data, [Online]. Available: http://www.w3schools.com/tags/ref_httpmethods.asp. [Zugriff am 22 04 2013].

[4] e. a. RFC 2616 Fielding, „HTTP/1.1: Response," w3, [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html#sec6. [Zugriff am 2013 04 23].

[5] J. Barrez, „The Activiti Performance Showdown," 28 06 2012. [Online]. Available: http://www.jorambarrez.be/blog/2012/06/28/the-activiti-performance-showdown/. [Zugriff am 23 04 2013].

[6] E. Börger, „Approaches to modeling business processes: a critical analysis of BPMN, workflow patterns and YAWL," *Software and Systems Modeling; ,* Bd. 11, Nr. 3, S. 305-316, 2012.

[7] A. Gadatsch, Grundkurs Geschäftsprozess-Management: Methoden und Werkzeuge für die IT-Praxis: Eine Einführung für Studenten und Praktiker, Wiesbaden: Vieweg+Teubner Verlag, 2012.

[8] T. Stahl und M. Völter, Modellgetriebene Softwareentwicklung. Techniken - Engineering - Management, Heidelberg: dpunkt.verlag GmbH, 2007.

[9] J. Trompeter und G. Pietrek, Modellgetriebene Softwareentwicklung. MDA und MDSD in der Praxis, Entwickler.Press, 2007.

[10] T. Rademakers, Activiti in Action. Executable business processes in BPMN 2.0, Manning Publications, 2012.

[11] P. Bertram und P. Schlinck, Ausbildungsliteratur. Private Kranken- und Pflegeversicherung, Karlsruhe: Verlag Versicherungswirtschaft, 2009.

[12] H. Eichenauer, P. Köster, V. Lüpertz und R. Schmalohr, Spezielle Versicherungslehre Band 2, Haan-Gruiten: Verlag Europa-Lehrmittel, 1999.

[13] „Activiti 5.12 User Guide," 12 April 2013. [Online]. Available: http://www.activiti.org/userguide/. [Zugriff am 12 April 2013].

[14] K. U. Bachmann, Maven 2. Eine Einführung, aktuell zur Version 2.0.9, München: Addison-Wesley Verlag, 2009.

[15] D. Heffelfinger, Java EE 6 with GlassFish 3 application server : a practical guide to install and configure the GlassFish 3 application server and develop Java EE 6 applications to be deployed to this server, Birmingham: Packt Publ., 2010.

[16] O. Corporation, „Java Naming and Directory Interface (JNDI)," Oracle Corporation, [Online].

Available: http://www.oracle.com/technetwork/java/jndi/index.html. [Zugriff am 24 04 2013].

[17] O. Corporation, „Java Transaction API (JTA),“ Oracle Corporation, [Online]. Available: http://www.oracle.com/technetwork/java/javaee/jta/index.html. [Zugriff am 24 04 2013].

[18] O. Corporation, „Introduction to the Java Persistence API,“ Oracle Corporation, [Online]. Available: http://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html. [Zugriff am 24 04 2013].

[19] O. Corporation, „Lesson: JDBC Introduction,“ Oracle Corporation, [Online]. Available: http://docs.oracle.com/javase/tutorial/jdbc/overview/index.html. [Zugriff am 24 04 2013].

[20] O. Corporation, „Class HttpServlet,“ Oracle Corporation, [Online]. Available: http://docs.oracle.com/javaee/1.4/api/javax/servlet/http/HttpServlet.html. [Zugriff am 25 04 2013].

[21] P. T. Eugster, P. A. Felber, R. Guerraoui und A.-M. Kermarrec, „http://www.eaipatterns.com/PublishSubscribeChannel.html,“ *ACM Computing Surveys,* Bd. 35, Nr. 2, S. 114-131, June 2003.

Abbildungsverzeichnis

Abbildung 1 Prozessmodell zur Veranschaulichung der BPMN 12
Abbildung 2 Top-Level-Prozessmodell Leistungsabrechnung 35
Abbildung 3 Sub-Prozess-Modell Belegerfassung 37
Abbildung 4 Sub-Prozess-Modell Belegkorrektur 38
Abbildung 5 Sub-Prozess-Modell Erstattungsberechtigung prüfen 38
Abbildung 6 Sub-Prozess-Modell Erstattungshöhe berechnen 39
Abbildung 7 Sub-Prozess-Modell Manuelle Korrekturen vornehmen 39
Abbildung 8 Technischer Aufbau 44
Abbildung 10 Fachliches Datenmodell 48
Abbildung 11 Weitere Beispielprozesse 54
Abbildung 12 Veranschaulichung des BPMN-Parsings 57
Abbildung 13 User-Task-Modell nach Erweiterung 60
Abbildung 14 Fehlerkategorien 61
Abbildung 15Prozessmodelle der verschiedenen Fälle für den Performance Test 63
Abbildung 16 Alternativer Sub-Prozess für die Belegerfassung 67

Tabellenverzeichnis

Tabelle 1 Eventtabelle 13
Tabelle 2 Activiti Listener 22
Tabelle 3 Activiti Listener Interfaces 23
Tabelle 4 Engine Interaktionen: Prozess starten & Task abschließen 25
Tabelle 5 Leistungsarten 29
Tabelle 6 Leistungsarten im Beispiel 40
Tabelle 7 Tarife im Beispiel 41
Tabelle 9 BPMN Konzepte & Elemente 51
Tabelle 10 Fehler im Prozess: Situationen, Quellen & Lösungen 62
Tabelle 11 Performancemessungen 64
Tabelle 12 Aufgabenstrukturierungsaspekte 68

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____ _____