HAW HAMBURG

MASTERTHESIS
Dennis Pietruck

Entwicklung und Evaluation eines Frameworks zur Generierung fachlicher Schnittstellen für Geschäftsprozesse

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Dennis Pietruck

Entwicklung und Evaluation eines Frameworks zur Generierung fachlicher Schnittstellen für Geschäftsprozesse

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Ulrike Steffens
Zweitgutachter: Prof. Dr. Martin Schultz

Eingereicht am: 20.04.2021

Dennis Pietruck

Thema der Arbeit

Entwicklung und Evaluation eines Frameworks zur Generierung fachlicher Schnittstellen für Geschäftsprozesse

Stichworte

Geschäftsprozesse, BPMN, REST, Prozessautomatisierung

Kurzzusammenfassung

Die Automatisierung von Prozessen mit Hilfe von Workflow Engines umfasst meist die implementierung von Schnittstellen für Prozessbeteiligte. Die Implementierten Schnittstellen sollen die Workflow Engine kapseln und die Anwendungsdomäne widerspiegeln. Ziel dieser Arbeit ist es, den Entwicklungsprozess für die fachliche Schnittstelle zu automatisieren, indem ein Framework entwickelt wird, welches auf Gundlage von Prozess- und Datenmodellen eine REST-Schnittstelle generiert. Dafür wird zunächst ein Metamodell entwickelt, mit dem Prozess- und Datenmodelle verknüpft werden können. Die Verknüpfung der unterschiedlichen Modelle bestimmt auch das Verhalten der generierten REST-Schnittstelle. Anschließend wird eine Architektur erarbeitet, mit der verschiedene Workflow Engines und HTTP-Bibliotheken mit dem Framework verwendet werden können. Zuletzt wird der Effekt des Frameworks auf Clientanwendungen anhand eines Beispiels untersucht.

Dennis Pietruck

Title of Thesis

Development and evaluation of a framework for generating domain-oriented interfaces for business processes

Keywords

Business Processes, BPMN, REST, Process Automation

Abstract

Automating processes with the help of workflow engines usually involves the implementation of APIs for process participants. The implemented interfaces should encapsulate the workflow engine and reflect the application domain. The aim of this work is to automate the development process for the domain-oriented interface by developing a framework that generates a REST interface based on process and data models. For this purpose, a meta model is first developed with which process and data models can be linked. The linking of the different models also determines the behaviour of the generated REST interface. Subsequently, an architecture is developed that allows different workflow engines and HTTP libraries to be used with the framework. Finally, the effect of the framework on client applications is evaluated by means of an example.

Inhaltsverzeichnis

Abbildungsverzeichnis **viii**

Tabellenverzeichnis **x**

Abkürzungen **xi**

1 Einleitung **1**
1.1 Motivation 1
1.2 Zielsetzung 2
1.3 Abgrenzung 3
1.4 Aufbau der Arbeit 4

2 Grundlagen **5**
2.1 Prozessautomatisierung 5
2.1.1 BPMN 5
2.1.2 Workflow Engines 10
2.2 Representational State Transfer 16

3 Analyse **19**
3.1 Problemstellung 19
3.1.1 Modellierung komplexer Daten mit BPMN 20
3.1.2 Modellierung von REST-Schnittstellen 20
3.1.3 Verknüpfung der einzelnen Modelle 21
3.1.4 Prozesse in Clientanwendungen 21
3.1.5 Verschiedene Produkte 21
3.2 Verwandte Arbeiten 22
3.2.1 Modellierung und Generierung von REST-Schnittstellen 22
3.2.2 Objektzentrierte Prozesse 23
3.2.3 Umsetzung von Prozessen mit REST-Schnittstellen 25

3.3 Anwendungsszenarien . . . 29
3.3.1 Frontend . . . 29
3.3.2 Verteilte Anwendungen . . . 30
3.3.3 API als Produkt . . . 30
3.4 Anforderungen . . . 31

4 Umsetzung **33**
4.1 Modellierung . . . 33
4.1.1 REST Ressourcen . . . 34
4.1.2 Modellverknüpfung . . . 37
4.1.3 Validierung von Ressourcen . . . 41
4.1.4 Metamodell . . . 42
4.2 Architektur . . . 44
4.2.1 Kontextabgrenzung . . . 45
4.2.2 Bausteinsicht . . . 45
4.2.3 Laufzeitsicht . . . 47
4.2.4 Verteilungssicht . . . 48
4.3 Implementierung . . . 50
4.3.1 Technologien . . . 50
4.3.2 Application Starter für erleichtertes Deployment . . . 51
4.3.3 Validierung von Ressourcen . . . 51
4.3.4 Stand der Umsetzung . . . 52

5 Evaluierung **54**
5.1 Beispielszenario . . . 54
5.1.1 Geschäftsprozess . . . 54
5.1.2 Datenmodell . . . 57
5.1.3 Umsetzung . . . 57
5.1.4 Implementierung mit generierter Schnittstelle . . . 59
5.1.5 Implementierung über die generische Schnittstelle der Workflow Engine . . . 59
5.2 Auswirkungen auf Clientanwendungen . . . 60
5.2.1 Wartbarkeit . . . 61
5.2.2 Effizienz . . . 63
5.2.3 Übertragbarkeit . . . 65
5.3 Diskussion . . . 66

6 Zusammenfassung und Ausblick **67**
6.1 Fazit 67
6.2 Ausblick 68

Literaturverzeichnis **70**

Selbstständigkeitserklärung **75**

Abbildungsverzeichnis

2.1 BPMN Aktivitäten 6
2.2 Blanko Start- Zwischen- und Endereignisse 7
2.3 Fangende und werfende Nachrichtenereignisse 7
2.4 Unterbrechende und nicht unterbrechende angeheftete Nachrichtenereignisse 8
2.5 Paralleles exklusives und ereignisbasiertes Gateways 8
2.6 Pool und Lanes 8
2.7 Zugeklappter Pool 9
2.8 Sequenz- und Datenfluss 9
2.9 Datenobjekt mit lesender und schreibender Verbindung 10
2.10 Datenspeicher mit lesender und schreibender Verbindung 10
2.11 Zustandsautomat für Aktivitäten [34] 12
2.12 Camunda Software Stack 14
2.13 jBPM Software Stack 15
2.14 Camunda Services 15

3.1 Beispiel für die Relation zwischen Prozessen und Daten [21] 24
3.2 Standard Lieferprozess 26
3.3 Lieferprozess mit kundenfreundlicher Stornierung 27

4.1 Beispielressource mit UML 35
4.2 Ressourcenbaum aus gerichteten Assoziationen 36
4.3 Verknüpfung von Ressource mit Datenobjekt 39
4.4 Prozess Antrag prüfen 40
4.5 Prozess Dokument anfordern 40
4.6 Metamodell 43
4.7 Kontextsicht 45
4.8 Laufzeit 47
4.9 Deployment Szenario: Framework und Engine in einer Datei 50
4.10 Deployment Szenario: Framework und Engine werden zur Laufzeit verknüpft 50

4.11 Deployment Szenario: Framework und Anwendung kommunizieren über REST 50
4.12 Validierung von Ressourcen mit dem RessourceController in der Implementierung 51

5.1 WP-Prozess 56
5.2 Datenmodell 58
5.3 Komponenten der Frontendanwendung 59
5.4 Komponenten der Frontendanwendung mit BPM Services 60
5.5 Berechnung der Strukturellen Systemkomplexität 61
5.6 Berechnung der McCabe Metrik 62
5.7 Ausschnitt aus dem Sonarqube Komplexitätsüberblick 63
5.8 Vergleich der Bearbeitungszeiten 64
5.9 Berechnung der Modulkopplung 65

Tabellenverzeichnis

5.1 Entwurfskomplexität für die Beispielanwendungen 62
5.2 Modulkopplung für die Beispielanwendungen 66

Abkürzungen

API application programming interface.

BPEL WS-Business Process Execution Language.

BPM Business Process Management.

BPMN Business Process Model and Notation.

BPMS Business Process Management System.

CRUD create, read, update, and delete.

HAL Hypertext Application Language.

HATEOAS Hypermedia as the Engine of Application State.

HTTP Hypertext Transfer Protocol.

ID Identifier.

OCL Object Constraint Language.

OMG Object Management Group.

REST Representational State Transfer.

RPC Remote Procedure Call.

UML Unified Modeling Language.

URI Uniform Resource Identifier.

WfMS Workflow Mangement System.

XML Extensible Markup Language.

1 Einleitung

Eine große Herausforderung für Unternehmen ist die effiziente Bewältigung komplexer, sich schnell ändernder Geschäftsprozesse. Typischerweise ist nicht nur eine Person oder eine Abteilung an der Prozessausführung beteiligt, da diese auch über Organisationsgrenzen hinweg stattfinden kann. Daher ist der Einsatz von Business Process Management (BPM) ein wichtiges Mittel zur Erfassung, Optimierung und Ausführung der Geschäftsprozesse und kann maßgeblich zum Erfolg eines Unternehmens beitragen [6].

Die Optimierung der Prozesse kann durch unterschiedliche Maßnahmen stattfinden. Ein häufig eingesetztes Mittel ist die Prozessautomatisierung. Ziel der Automatisierung ist eine effiziente und einheitliche Ausführung von Geschäftsprozessen zu ermöglichen. Dabei können einzelne Prozessschritte, oder sogar der gesamte Prozess, durch Informationssysteme ausgeführt werden. Systeme, die den gesamten Prozesszusammenhang berücksichtigen und verwalten, können eine Organisation unterstützen, die Komplexität der Geschäftsprozesse zu bewältigen und deren Effektivität zu steigern [38]. Häufig eingesetzte Systeme sind Business Process Management Systeme (BPMSs), welche typischerweise mit einem Prozessmodellierungswerkzeug, einer Workflow Engine zur Ausführung des Prozesses, sowie einer Aufgabenverwaltung für Prozessbearbeiter ausgestattet sind [19].

1.1 Motivation

Moderne Workflow Engines steuern die Prozessausführung auf Grundlage von Prozessmodellen und können als Bibliothek in Anwendungen integriert werden. Eine häufig eingesetzte Modellierungssprache ist die Business Process Model and Notation (BPMN), welche es ermöglicht, Parallelität sowie bedingte Ausführung von Aktivitäten abzubilden. Weiterhin kann mit dieser Modellierungssprache eine feine Unterscheidung von

Aktivitäts- und Ereignisarten getroffen werden. Darüber hinaus wurde für die Modellierungssprache nicht nur eine Spezifikation für die graphischen Elemente entwickelt, sondern auch ein standardisiertes, maschinenlesbares Datenformat für Prozessmodelle festgelegt. Dadurch können Modelle, die BPMN-konform sind, von jeder BPMN-konformen Engine ausgeführt werden.

Typischerweise bieten die Engines Schnittstellen, welche die Steuerung von Prozessinstanzen und der Engine ermöglichen. Neben einer Programmiersprachenschnittstelle wird auch häufig eine REST API angeboten, welche für verteilte Anwendungsszenarien verwendet wird. In modernen Unternehmen ist REST als Architekturstil weit verbreitet, da dieser lose Kopplung zwischen einzelnen Komponenten fördert und leicht mit den standard-HTTP Bibliotheken gängiger Programmiersprachen implementiert werden kann.

Häufig werden für Process Engines Benutzeroberflächen für Prozessbeteiligte entwickelt, oder Schnittstellen implementiert, welche von anderen Services genutzt werden. Diese Services können Teil der eigenen Anwendungslandschaft sein, oder auch zu anderen Organisationen gehören. Letzteres kann vor allem dann der Fall sein wenn der Prozess über Organisationsgrenzen hinweg stattfindet. Hierbei könnte zwar direkt die REST-Schnittstelle der Engine verwendet werden, um die Entwicklungsdauer und Kosten zu reduzieren, allerdings entstehen dadurch auch einige Nachteile bei der Entwicklung von Systemen, welche die Schnittstelle nutzen. Für die Nutzung der generischen Schnittstelle der Workflow Engine benötigen Entwickler Wissen über den Geschäftsprozess, die BPMN, sowie über die Engine. Weiterhin steigt die Kopplung zwischen Engine und den Schnittstellennutzern [37].

1.2 Zielsetzung

Im Rahmen dieser Masterarbeit soll eine Lösung erarbeitet werden, mit der der Aufwand für die Entwicklung von fachlichen Schnittstellen für Prozessanwendungen reduziert wird. Dazu wird ein Framework entwickelt, welches eine domänenorientierte REST-Schnittstelle auf Grundlage von Prozess- und Datenmodellen für Anwendungen, die auf einer Workflow Engine basieren, generiert.

Für die Modellierung von REST-Ressourcen existiert keine standardisierte Notation. Daher muss für das zu entwickelnde Framework eine Modellierungssprache für REST-Ressourcen entwickelt werden.

Im Gegensatz zu Prozessmodellen bieten REST-Schnittstellen eine objektorientierte Sicht auf ein System. Daher soll ein Konzept erarbeitet werden, mit dem die unterschiedlichen Sichten zusammengeführt werden können. Zum einen muss eine Verknüpfungsmethode für die Prozess- und Ressourcenmodelle geschaffen werden, zum anderen muss für diese Verknüpfung eine Semantik entwickelt werden, welche das Laufzeitverhalten der modellierten und verknüpften Komponenten beschreibt.

1.3 Abgrenzung

Für die Modellierung von Prozessen existieren verschiedene Notationen wie die BPMN oder die WS-Business Process Execution Language (BPEL). Da moderne Workflow Engines häufig die BPMN verwenden, beschränkt sich die Lösung auf die BPMN für die Prozessmodellierung.

Darüber hinaus behandelt die Arbeit lediglich REST-Schnittstellen als Kommunikationsmittel, da diese etabliert sind und sich bereits stark durchsetzen konnten. Außerdem bieten Workflow Engines bis jetzt nur REST-Schnittstellen an. Damit ist das zu entwickelnde Framework als Ergänzung zu den bereits vorhanden Schnittstellen zu verstehen.

Andere Schnittstellenarten wie GraphQL bieten mit den Queries und Mutations sehr mächtige Abfragesprachen. In [14] wird gezeigt, dass diese auch für die Kommunikation mit Workflow Engines nützlich sind. Allerdings können einzelne Abfragen komplex sein und werden vor allem dann eingesetzt, wenn eine Anwendung mit vielen stark vernetzten Datenobjekten arbeitet. Ein Geschäftsprozess verwaltet zwar auch unterschiedlichste Datenobjekte, allerdings sind diese meist auf eine Prozessausführung beschränkt und selten mit vielen weiteren Objekten verknüpft. Die in [14] gezeigten Vorteile können ebenfalls durch die Verwendung fachlicher Daten erreicht werden.

In [31] wird eine Lösung vorgestellt, bei der REST-Schnittstellen für BPMN Choreographien generiert werden. Dabei liegt die Anwendung zwischen den Prozessbeteiligten und koordiniert ihre Kommunikation. Diese Arbeit befasst sich nicht mit Choreographiemodellen, da das Ziel des Frameworks die Abstraktion der generischen Schnittstelle der

Workflow Engine ist. Die zur Verfügung gestellte Schnittstelle muss nicht zwangsläufig in Choreographien eingesetzt werden.

1.4 Aufbau der Arbeit

Die Arbeit besteht aus fünf Kapiteln. Anschließend an die Einführung werden in Kapitel zwei die Grundlagen für das Thema der Arbeit, REST-Schnittstellen sowie Prozessautomatisierung mit der BPMN, vorgestellt.

Das dritte Kapitel befasst sich mit der Analyse der Problemstellung mit dem Ziel, Anforderungen an das zu entwickelnde Framework zu formulieren. Dazu werden verwandte Arbeiten untersucht sowie mögliche Anwendungsszenarien dargestellt.

Mit den gesammelten Anforderungen wird dann, in Kapitel 4, die Umsetzung des Frameworks beschrieben. Dazu wird das dem Framework zugrundeliegende Metamodell erläutert, die Architektur des Frameworks und einige Implementierungsdetails präsentiert.

Das entwickelte Framework wird in Kapitel 5 evaluiert. Dazu wird eine Beispielanwendung mit dem Framework umgesetzt, um Auswirkungen auf die Codequalität zu messen.

Zuletzt werden die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick auf weiterführende Arbeiten gegeben.

2 Grundlagen

Im folgenden Kapitel werden die für diese Arbeit relevanten Grundlagen erläutert. Dazu wird die Modellierungssprache BPMN vorgestellt und erklärt, wie diese mit Hilfe von Workflow Mangement Systemen (WfMS) für die Automatisierung von Prozessen genutzt werden kann. Anschließend werden in 2.2 die wichtigsten Eigenschaften des REST Architekturstils erläutert.

2.1 Prozessautomatisierung

Prozessautomatisierung kann in verschiedenen Abstufungen stattfinden. In der einfachsten Umsetzung werden einzelne Aktivitäten eines Geschäftsprozesses durch Informationssysteme ausgeführt. Dabei liegen in den einzelnen Systemen keine Informationen über den Geschäftsprozess vor, an dem sie beteiligt sind. Ein Mailsystem kann beispielsweise automatisch E-Mails versenden, hat aber keine Informationen über den Geschäftsprozess, in dem der Versand stattfindet. Ein anderer Ansatz ist die Automatisierung durch Systeme, welche die Geschäftsprozesse kennen und steuern. Dies kann zum Beispiel durch Workflow Engines geschehen [9].

Doch bevor ein Prozess automatisiert ausgeführt werden kann, muss dieser erst ermittelt, beziehungsweise entworfen und festgehalten werden. Für die Dokumentation von Prozessen wurden verschiedene Modellierungssprachen entwickelt. Aktuell weit verbreitet ist die BPMN, welche auch von vielen Workflow Engines unterstützt wird [12].

2.1.1 BPMN

Die BPMN ist eine von der Object Management Group (OMG) standardisierte Modellierungssprache für Geschäftsprozesse. Die Spezifikation beschreibt die graphischen Elemente der Sprache und deren Ausführungssemantik, sowie ein auf der Extensible

Markup Language (XML) basierendes, maschinenlesbares Datenformat. Durch die Verwendung der Sprache soll es ermöglicht werden, schnell und einfach Modelle zu entwickeln und dabei trotzdem komplexe Prozesse abzubilden. Diese sich entgegenstehenden Anforderungen werden ermöglicht, indem die Modellierungselemente in die fünf Kategorien Flussobjekte, verbindende Objekte, Teilnehmer, Artefakte und Daten aufgeteilt sind. Innerhalb dieser Kategorien befinden sich leicht verständliche Basiselemente, welche für die Modellierung gängiger Geschäftsprozesse ausreichend sind, aber auch spezifische, von den Basiselementen abgeleitete, Elemente für speziellere Anwendungsfälle [34]. Im Folgenden wird ein kurzer Überblick über die einzelnen Kategorien gegeben sowie einzelne Elemente kurz erläutert.

Flussobjekte

Flussobjekte beschreiben das Verhalten eines Prozesses. Hierbei wird unterschieden zwischen Aktivitäten, Gateways und Ereignissen.

Aktivitäten stellen Tätigkeiten dar, die während des Prozessablaufs erledigt werden. Der genaue Aktivitätstyp wird durch weitere Symbole, die der Aktivität angehängt werden, weiter spezifiziert. Beispiele für Aktivitäten sind das Versenden von E-Mails, Nutzereingaben oder von Informationssystemen ausgeführte Aufgaben. Neben diesen atomaren Aktivitäten, also Aufgaben, die nicht weiter in einzelne Schritte aufgeteilt werden, gibt es noch Aktivitäten, die durch einen eigenen Prozess dargestellt werden. *Call Activites* stellen Aktivitäten innerhalb eines Prozesses dar, die einen anderen Prozess aufrufen. Subprozesse sind ebenfalls Aktivitäten, die durch einen Prozess definiert werden. Allerdings ist hierbei die Kopplung zwischen den einzelnen Prozessen höher als bei der Verwendung von *Call Activities*.

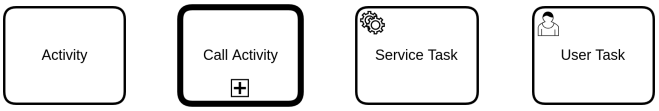


Abbildung 2.1: BPMN Aktivitäten

Ereignisse werden verwendet, um darzustellen, dass während eines Prozessablaufs etwas Wichtiges geschehen ist. Dabei wird zwischen Startereignissen, Zwischenereignissen und Endereignissen unterschieden. Durch die Verwendung weiterer Symbole lässt sich bestimmen, was genau während des Ereignisses geschehen ist. So lässt sich beispielsweise

modellieren, dass eine Nachricht empfangen wurde, dass ein bestimmtes Datum erreicht wurde oder dass während der Prozessausführung ein Fehler aufgetreten ist.

Abbildung 2.2: Blanko Start- Zwischen- und Endereignisse

Ereignisse, die während des Prozessablaufs stattfinden, werden in die Kategorien werfende, fangende und angeheftete Ereignisse unterteilt. Werfende Ereignisse werden durch den fortschreitenden Prozessablauf ausgelöst, damit andere Prozessbeteiligte darauf reagieren können. Fangende Ereignisse werden durch Prozessbeteiligte ausgelöst. Der Prozessablauf kann auf diese Ereignisse explizit warten, bevor die Ausführung an einer bestimmten Stelle fortgesetzt wird.

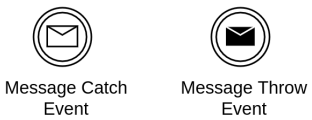


Abbildung 2.3: Fangende und werfende Nachrichtenereignisse

Angehängte Ereignisse sind mit Aktivitäten verknüpft und werden verwendet, um die Ereignisse, die während der Aktivitätsausführung stattfinden können, zu modellieren. Diese können von der Aktivität selbst ausgelöst werden, zum Beispiel bei Fehlerereignissen, oder von anderen Prozessteilnehmern, zum Beispiel durch den Empfang einer Nachricht. Wenn das Ereignis die Aktivitätsausführung unterbrechen soll, werden zur Modellierung unterbrechende Ereignisse verwendet. Hierbei wird der Prozessablauf an einem alternativen Pfad fortgesetzt. Nicht unterbechende Ereignisse haben keinen Einfluss auf die Aktivität. Allerdings kann hierdurch die Ausführung eines parallelen Prozessablaufs ausgelöst werden.

Gateways werden verwendet, um den Prozessfluss zu steuern. Dadurch lassen sich bedingte oder parallele Prozessausführungen modellieren. Bedingungen können Aussagen sein, die für Prozessdaten gelten müssen. Dann handelt es sich um ein datenbasiertes Gateway. Das Eintreten eines Ereignisses kann ebenfalls als Bedingung verwendet werden. Dann handelt es sich um ein ereignisbasiertes Gateway.

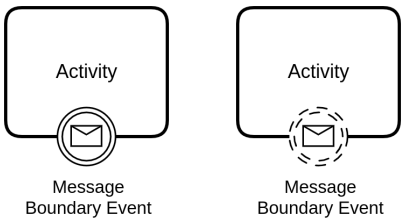


Abbildung 2.4: Unterbrechende und nicht unterbrechende angeheftete Nachrichtenereignisse

Abbildung 2.5: Paralleles exklusives und ereignisbasiertes Gateways

Teilnehmer

Teilnehmer dienen der Strukturierung von Prozessen. Dabei können Prozesse auf verschiedene Prozessteilnehmer aufgeteilt werden. Ein Prozessteilnehmer wird durch einen Pool dargestellt. Ein Teilnehmer kann beispielsweise eine einzelne Person, ein Unternehmen oder eine andere Art von Organisation sein. Um eine feinere Zuordnung von Aktivitäten zu machen, können Pools durch Lanes aufgeteilt werden. Diese repräsentieren Untereinheiten eines Pools.

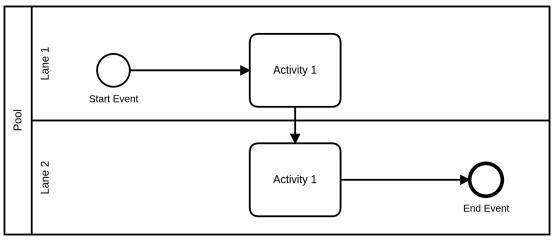


Abbildung 2.6: Pool und Lanes

Bei der Modellierung von Prozessen sind nicht immer alle Abläufe der anderen Prozessteilnehmer bekannt. Damit sich diese Teilnehmer trotzdem darstellen lassen, können Pools auch zugeklappt werden. Ein zugeklappter Pool trägt nur den Namen des Prozessteilnehmers und zeigt nicht, was innerhalb des Pools ausgeführt wird.

Abbildung 2.7: Zugeklappter Pool

Verbindende Objekte

Verbindende Objekte beschreiben die Beziehungen zwischen Flussobjekten und Teilnehmerobjekten. Um die Ausführungsreihenfolge von Flussobjekten innerhalb eines Pools zu modellieren, werden Sequenzflüsse verwendet. Da jeder Prozessteilnehmer seine eigenen Prozesse selbst steuert, können Sequenzflüsse nicht über Poolgrenzen hinweg verlaufen. Dies würde bedeuten, dass verschiedene Prozessteilnehmer ihre Prozesse gemeinsam steuern. Um die Verbindungen zwischen den einzelnen Prozessteilnehmern darstellen zu können, werden Nachrichtenflüsse verwendet. Typischerweise werden während der Ausführung von Prozessen mit mehreren Teilnehmern Nachrichten ausgetauscht, die ein Ereignis beim Empfänger auslösen und dadurch den Prozess beim Teilnehmer vorantreiben. Daher werden für die Modellierung von Prozessabläufen über Organisationsgrenzen hinweg, dargestellt durch Pools, Nachrichtenflüsse verwendet.

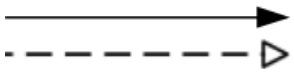
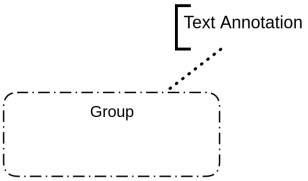


Abbildung 2.8: Sequenz- und Datenfluss

Artefakte

Artefakte haben keinen Einfluss auf den Prozessablauf, sondern werden verwendet, um Prozesse mit weiteren Informationen für besseres Verständnis oder zur Dokumentation zu ergänzen. Mit Gruppen können Teile eines Prozessmodells zusammengefasst werden. Text-Annotationen dienen dazu, Prozessmodelle mit Kommentaren zu versehen.



Daten

Während der Ausführung eines Geschäftsprozesses werden häufig Daten gelesen oder erzeugt. Um diese im Modell darzustellen, werden Datenobjekte verwendet. Dabei ist nicht festgelegt, um was für Daten es sich genau handelt. So kann es sich bei Datenobjekten zum Beispiel um physische Dokumente, Benutzereingaben oder einzelne Dateien handeln. Um festzuhalten, welche Aktivität ein Datenobjekt liest, erzeugt oder verändert, werden Datenassoziationen verwendet. Diese sind durch einen Pfeil dargestellt, dabei macht die Pfeilrichtung deutlich, ob eine Aktivität Daten liest oder schreibt.

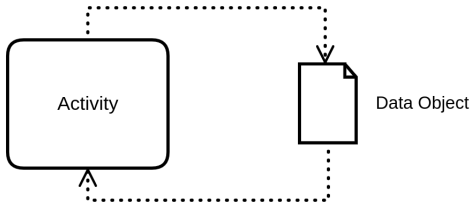


Abbildung 2.9: Datenobjekt mit lesender und schreibender Verbindung

Neben einzelnen Datenobjekten lassen sich mit der BPMN auch Datenspeicher darstellen. Auch hier ist durch die BPMN nicht festgelegt, was ein Datenspeicher genau ist. So kann dieser beispielsweise eine Datenbank, eine bestimmte Software oder Phyisches wie ein Fahrtenbuch sein. Diese Dinge existieren unabhängig von einzelnen Prozessinstanzen, können aber durch ihre Aktivitäten gelesen und manipuliert werden.

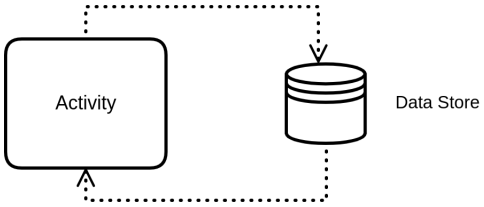


Abbildung 2.10: Datenspeicher mit lesender und schreibender Verbindung

2.1.2 Workflow Engines

Vereinfacht ausgedrückt sind Workflow Engines Interpreter für Prozessmodelle. Sie führen Geschäftsprozesse auf Grundlage eines Modells von Anfang bis Ende aus. Häufig sind Engines Teil eines WfMS oder eines BPMS. Eine Abgrenzung der Begriffe ist schwierig, da es eine große Schnittmenge in den Funktionalitäten gibt. Jedoch sind Funktionalitäten,

welche das Business Process Management unterstützen sollen, wie Prozessanalyse und Monitoring, ein Hinweis dafür, dass es sich um ein BPMS handelt. In dieser Arbeit werden die Begriffe als Synonym verwendet, da die gemeinsame Komponente, die Process Engine, betrachtet wird.

Durch den Einsatz solch einer Engine soll eine Entkopplung von Geschäftsprozess und Anwendungscode stattfinden, da die Prozesslogik nicht im Programmcode einer Anwendung festgeschrieben ist, sondern im Modell abgebildet und von der Engine ausgeführt wird. Welche Modelle ausgeführt werden können, ist von der Engine abhängig. Aktuelle Engines unterstützen meist die offene, standardisierte BPMN, weshalb sich diese Arbeit auch auf diese Modelle beschränkt. Trotzdem sind Engines, welche sich auf die Ausführung proprietärer Modelle beschränken, nichts ungewöhnliches [25].

Prozessausführung

Damit eine Engine Prozesse ausführen kann, benötigt diese eine *Prozessdefinition*. Hierbei handelt es sich um ein Prozessmodell, welches mit den für die Ausführung benötigten Informationen versehen ist. Das sind zum Beispiel Referenzen zu anderen Prozessdefinitionen bei *Call Activities* oder Skripte für *Script Tasks*. Für jeden gestarteten Prozess wird von der Engine eine *Prozessinstanz* erzeugt. Diese repräsentiert genau eine Prozessausführung. Der Zusammenhang zwischen Prozessinstanz und Prozessdefinition ist vergleichbar mit dem Verhältnis zwischen Objekt und Klasse aus der objektorientierten Programmierung.

Um den Verlauf einer Prozessinstanz beschreiben zu können, wird in der BPMN-Spezifikation das Token-Konzept eingeführt, welches den Marken aus der Modellierung mit Petrinetzen ähnelt. Ein Token beschreibt während der Prozessausführung an welcher Stelle sich eine Prozessinstanz befindet. Mit dem Beenden einer Aktivität verschiebt sich das Token zur nächsten ausgeführten Aktivität. Innerhalb einer Prozessinstanz können auch mehrere Token vorkommen. Nach einem parallelen Gateway teilt sich ein Token auf, und symbolisiert damit die gleichzeitige Ausführung der mit dem Token markierten Aktivitäten.

Wenn ein Token an einer Aktivität angelangt ist, nimmt diese verschiedene Zustände an. Diese sind in der Spezifikation durch den in Abbildung 2.11 dargestellten Automaten beschrieben.

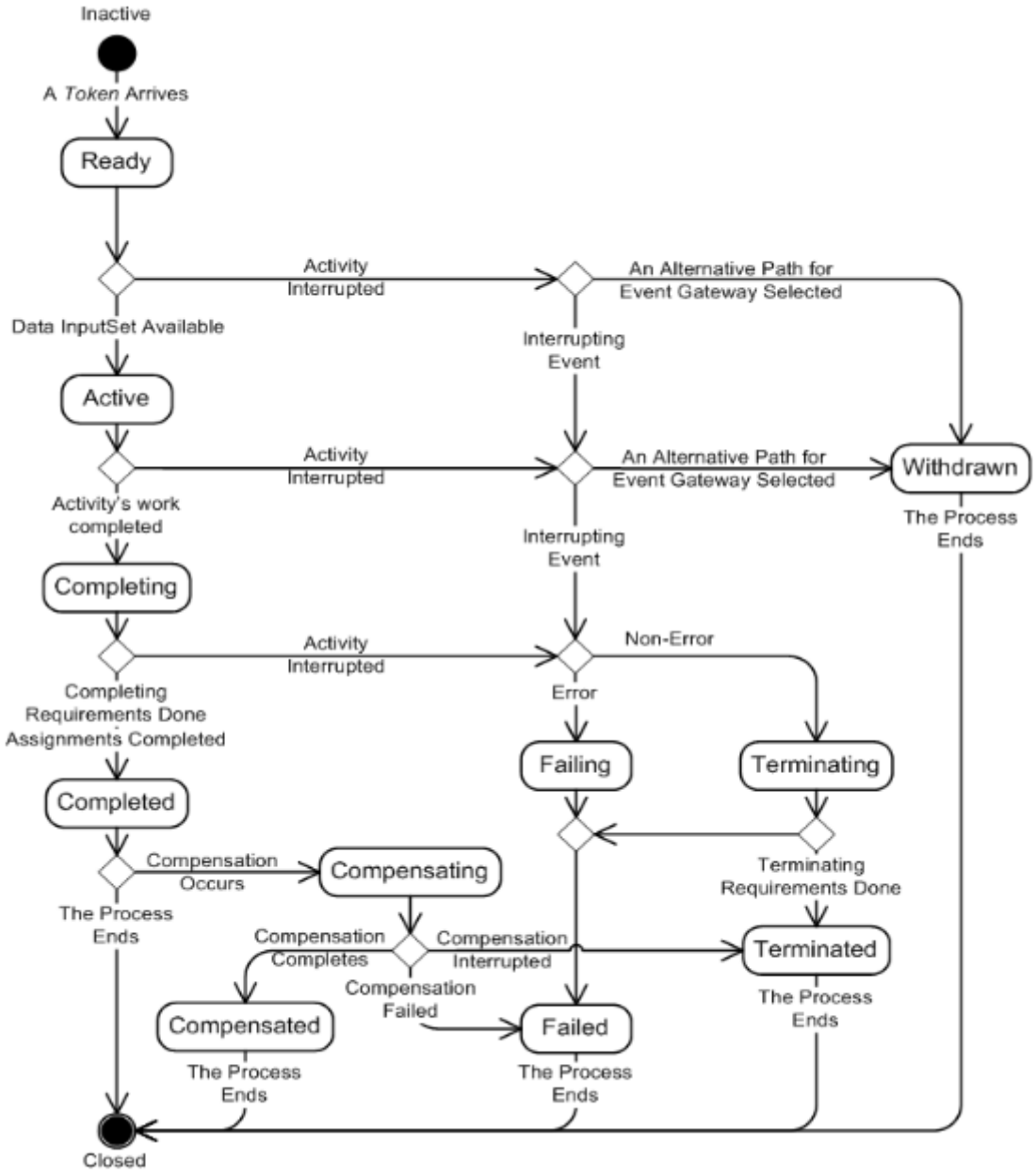


Abbildung 2.11: Zustandsautomat für Aktivitäten [34]

Wenn eine neue Prozessinstanz erzeugt wurde, sind alle dazugehörigen Aktivitäten zunächst im Startzustand *inaktiv*. Sobald ein Token an einer Aktivität angelangt ist, wechselt diese in den Zustand *bereit*. Von dort wechselt die Aktivität in den Zustand *aktiv*, sobald alle mit der Aktivität verknüpften Datenobjekte verfügbar sind. Wenn dies der Fall ist, ist die Aktivität *aktiv*. Hier findet die eigentliche Ausführung statt. Bevor die Ak-

tivität komplett abgeschlossen ist, wechselt sie in den Zustand *abschließen*. Hier werden alle Ereignisse, die mit der Aktivität verknüpft sind, ausgeführt. Wenn dies beendet wurde, wechselt die Aktivität in den Zustand *abgeschlossen* und von dort in den Endzustand *beendet*. Bevor eine Aktivität abgeschlossen ist, kann sie jederzeit durch ein Ereignis unterbrochen werden. Wenn es sich um ein unterbrechendes Ereignis handelt, wechselt die Aktivität in den Zustand *gescheitert* oder *beendet*, abhängig davon, ob es sich um ein Fehlerereignis handelt oder nicht. Wenn die Aktivität mit einem ereignisbasierten Gateway verknüpft ist, kann sie aus dem Zustand *bereit* und *aktiv* auch in den Zustand *zurückgezogen* wechseln, wenn ein anderes nicht unterbrechendes Ereignis des Gateways ausgelöst wurde. Eine beendete Aktivität kann ebenfalls noch unterbrochen werden. Wenn eine Kompensation auftritt, wechselt diese in den Zustand *kompensieren*. Die Kompensation kann erfolgreich abgeschlossen werden, scheitern oder durch ein Ereignis unterbrochen werden. Abhängig davon wechselt die Aktivität in den Zustand *kompensiert*, *gescheitert* oder *terminiert*, bevor sie geschlossen wird.

Prozessdaten

Die BPMN ermöglicht durch die Verwendung von Datenobjekten (siehe 2.1.1) die Darstellung von Daten, die während der Prozessausführung benötigt werden. Dadurch lassen sich außerdem Abhängigkeiten zwischen Daten und einzelnen Aktivitäten visualisieren. Feinere Beschreibungen der Daten durch beispielsweise Kompositionen oder Vererbung sind nicht möglich. Für die Steuerung des Prozessflusses werden die Flusselemente verwendet, für die eine Verbindung mit Datenelementen nicht zwangsweise stattfinden muss, auch wenn einige Gateways den Prozessfluss auf Grundlage von Prozessdaten steuern [27].

Einige Engines führen, um Prozessdaten verwenden zu können, das Konzept der Prozessvariable ein. Diese können beliebige Daten speichern, sowohl primitve Datentypen, als auch komplexe Objekte. Prozessvariablen werden nicht visuell durch ein eigenes Symbol festgehalten, sondern typischerweise durch herstellereigene XML Elemente realisiert. Datenelemente werden hierbei lediglich für Dokumentationszwecke verwendet und haben keinen Einfluss auf die Prozessausführung [3, 2, 1].

Zur Laufzeit kann eine Prozessinstanz mehrere Variablen erzeugen. Da die einzelnen Prozessinstanzen voneinander isoliert ausgeführt werden, können sie auch nicht auf die

Variablen anderer Instanzen zugreifen. Eine Variable gehört also zu genau einer Prozessinstanz. Die Daten der Prozessvariablen können zur Steuerung des Prozessablaufs auch an datenbasierten Gateways verwendet werden. Die ausgehenden Kanten eines Gateways müssen dazu mit Aussagen über die Daten annotiert werden, die gelten müssen, damit der annotierte Pfad ausgeführt wird.

Produkte

Bekannte Open Source BPMN Engines sind Activity[1], Camunda[2], und jBPM[3]. Neben den Engines selbst bestehen die Produkte meist aus weiteren Werkzeugen und Systemen, welche mit der Engine für Prozessautomatisierung verwendet werden können. Ein Vergleich der Produktübersichten von Camunda (Abbildung 2.12) und jBPM (Abbildung 2.13) macht dies deutlich. Beide Systeme bieten ein Werkzeug zur Modellierung von Ge-

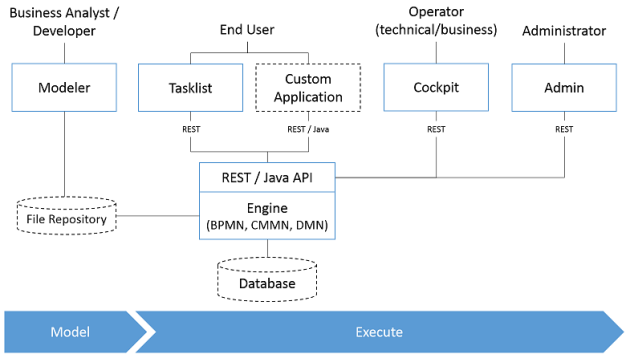


Abbildung 2.12: Camunda Software Stack

schäftsprozessen, welches die Modelle im BPMN konformen Standard speichert. Darüber hinaus wird eine Task List geboten, mit der Prozessbearbeiter Human Tasks ausführen können. Beide Produkte bieten eine Komponente für das Prozessmonitoring. Diese ist bei jBPM unter dem Namen Business Activity Monitoring zu finden. Die bei Camunda äquivalente Komponente ist das Cockpit.

Weitere Ähnlichkeiten sind auch bei dem Zugriff auf die Engine selbst zu finden. Alle Produkte bieten eine REST und eine Java-basierte Schnittstelle, welche von den mitgelieferten Komponenten verwendet werden, aber auch für die Einbettung in eigene Systeme

[1] https://www.activiti.org/
[2] https://camunda.com/
[3] https://www.jbpm.org/

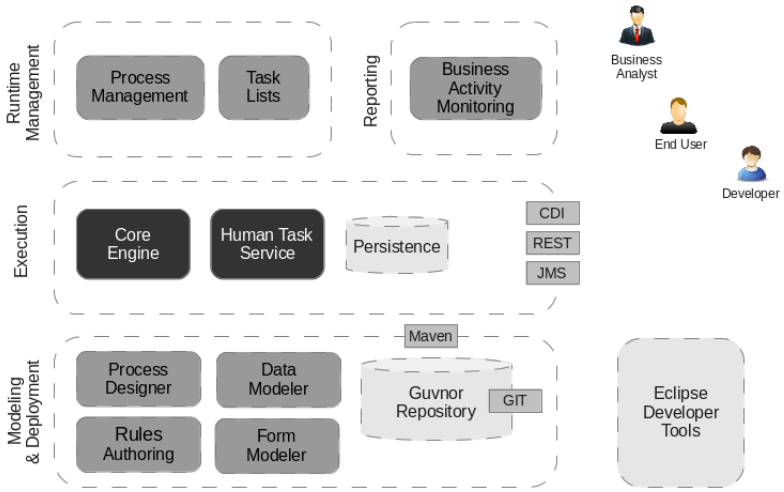


Abbildung 2.13: jBPM Software Stack

verwendet werden können. Bei einem Vergleich der Java-Schnittstellen fällt auf, dass auch hier eine ähnliche Architektur vorliegt. Abbildung 2.14 gibt eine Übersicht über einige von der Camunda Engine angebotenen Schnittstellen. Sowohl die Camunda als auch die jBPM Engine bieten Schnittstellen für die Steuerung einzelner Prozessinstanzen und Aktivitäten, zur Verwaltung von Prozessmodellen und zur Manipulation von Prozessdaten.

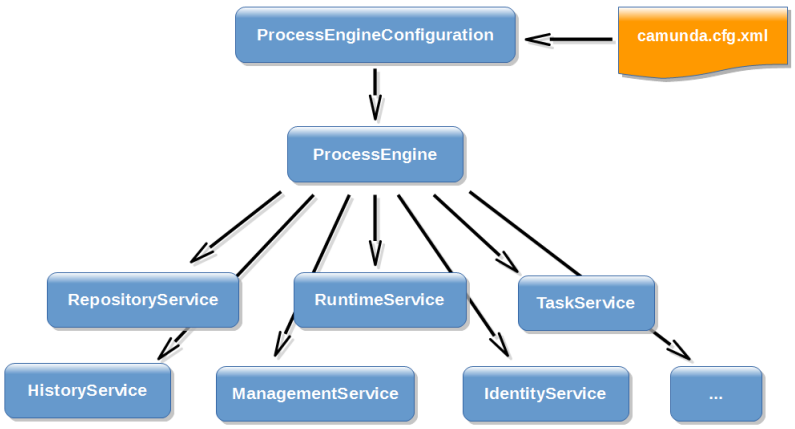


Abbildung 2.14: Camunda Services

2.2 Representational State Transfer

REST ist ein Architekturstil für verteilte Systeme, der die Struktur des World Wide Web abstrahiert und die Grundlage des HTTP bildet. Roy Fielding, der zur Spezifikation des Protokolls beigetragen hat [30], hielt die dafür geltenden Prinzipien im Rahmen seiner Dissertation unter dem Namen REST fest. Die beschriebenen Prinzipien beziehen sich auf die Kommunikation, also vor allem die Schnittstellen zwischen einzelnen Komponenten einer Architektur. Daher werden Schnittstellen, die sich an den Architekturstil halten, auch REST APIs genannt. Insgesamt werden sechs Bedingungen beschrieben, die für ein System, welches REST umsetzt, gelten sollen. Viele davon, wie Client-Server oder zustandslose Kommunikation, finden sich auch in anderen Architekturen. Was REST von anderen Architekturstilen unterscheidet, ist die Bedingung, dass alle Komponenten über eine einheitliche Schnittstelle kommunizieren (*Uniform Interface*). Für diese Schnittstellen wurden vier weitere Bedingungen aufgezählt, die erfüllt werden müssen [11]. Diese werden im Folgenden erläutert.

Identifizierbarkeit von Ressourcen

Ressourcen sind das Kernelement von REST APIs. Hierbei handelt es sich um Objekte mit Attributen, die einen Typ haben können und in Relation zueinander stehen. Eine Bedingung für die einheitliche Schnittstelle ist, dass jede Ressource durch eine URI identifiziert wird. Diese URI wird auch verwendet, um eine Ressource zu lesen oder zu bearbeiten.

Manipulation von Ressourcen durch ihre Repräsentation

Wenn eine Ressource gelesen wird, versendet der Server lediglich eine Repräsentation, die unabhängig von der Darstellung auf dem Server ist. Wenn eine Ressource bearbeitet werden soll, führt der Client die benötigten Änderungen selber aus und sendet die manipulierte Repräsentation an den Server. Dieser Mechanismus, Manipulation von Ressourcen durch ihre Repräsentation, ist eine weitere Bedingung des REST-Architekturstils.

Selbstbeschreibende Nachrichten

Weiterhin muss für die Schnittstellen gelten, dass selbstbeschreibende Nachrichten ausgetauscht werden. Das heißt, dass eine Nachricht alle Informationen enthält, die benötigt werden, damit ein Server weiß, wie er diese zu verarbeiten hat. Dazu können Nachrichten auch mit Metadaten angereichert werden.

Hypermedia as the Engine of Application State

Zuletzt gilt, dass HATEOAS umgesetzt wird. Hierbei werden Serverantworten mit Hyperlinks ergänzt, die auf Ressourcen zeigen, die mit der angefragten Ressource verknüpft sind. Weiterhin können so die auf einer Ressource zulässigen Operationen mitgeteilt werden. Dies soll ermöglichen, dass eine Schnittstelle dynamisch verwendet werden kann, ohne dass ein Entwickler alle Endpunkte im Vorfeld kennt. Weiterhin können so auch Änderungen an der Schnittstelle vorgenommen werden, ohne dass der Client angepasst werden muss.

REST Reifegrade

Auf den Bedingungen, die für REST-Schnittstellen gelten sollen, basiert auch das Richardson Maturity Modell. Durch dieses Modell lässt sich bewerten, wie gut eine HTTP basierte Schnittstelle die REST Prinzipien umsetzt. Das Modell ordnet Schnittstellen in eine von vier Stufen ein. Eine Schnittstelle, die alle REST Prinzipien einhält, wird dabei in die höchste Stufe eingeordnet.

In der ersten Stufe verwendet eine Schnittstelle das HTTP-Protokoll lediglich zum Nachrichtenaustausch. Es werden keine Mechanismen wie HTTP-Methoden, verschiedene URLs zur Identifizierung von Ressourcen, oder Statuscodes verwendet. Dadurch entsteht eine auf HTTP basierte Remote Procedure Calls (RPCs) Implementierung.

In der zweiten Stufe werden Uniform Resource Identifiers (URIs) verwendet, um einzelne Ressourcen zu unterscheiden. Weiterhin werden die typischen HTTP-Methoden POST und GET verwendet, um auf Ressourcen zuzugreifen beziehungsweise diese zu manipulieren.

Wenn eine Schnittstelle die Bedingungen für die ersten zwei Stufen erfüllt und darüber hinaus die HTTP-Methoden richtig verwendet, wird diese in die dritte Stufe eingeordnet.

Die richtige Verwendung sieht zum Beispiel vor, dass Aufrufe mit dem POST-Verb nicht idempotent sind. Daher werden POST-Aufrufe meist für die Erzeugung neuer Ressourcen oder für einen einmalig möglichen Zustandswechsel verwendet. Wenn idempotente Aufrufe implementiert werden sollen, müssen PUT oder PATCH Anfragen verwendet werden.

Die letzte Stufe des Richardson Maturity Modells ist dann erreicht, wenn eine Schnittstelle HATEOAS umsetzt. Wie eine Serverantwort mit Hyperlinks ergänzt wird, ist nicht vorgeschrieben. So können Links in Metadaten oder direkt der angefragten Ressourcenrepräsentation hinzugefügt werden. Ein Vorschlag, der die Ressourcenrepräsentation verwendet, wird in [18] beschrieben. Hierbei soll in allen Ressourcen, das Feld "_links" verwendet werden, um Hyperlinks für die Umsetzung von HATEOAS übergeben zu können. Dieser Mechanismus kann unabhängig vom Datenformat für die Repräsentation der Daten verwendet werden. Andere Verfahren wie ATOM [32] sind zwar als RFC standardisiert, beschränken sich allerdings auf ein spezielles Datenformat.

3 Analyse

In diesem Abschnitt werden die Herausforderungen beschrieben, die für die Entwicklung des Frameworks zur Generierung von Schnittstellen relevant sind. Dazu wird die Problemstellung genauer betrachtet, und es werden die sich daraus ergebenden Teilaspekte untersucht.

Anschließend werden verwandte Arbeiten und Projekte vorgestellt. Dabei wird eine Abgrenzung zu dieser Arbeit aufgezeigt, indem die Anwendbarkeit der vorgestellten Projekte auf die Problemstellung dieser Arbeit diskutiert wird. Zum anderen werden Gemeinsamkeiten zwischen den Problemstellungen und Lösungsansätzen der Arbeiten gefunden die zur Entwicklung des Frameworks beitragen.

Weiterhin werden in diesem Kapitel Anwendungsszenarien für das Framework vorgestellt. Hierbei werden die Besonderheiten der einzelnen Szenarien herausgearbeitet, aus denen Anforderungen an das Framework abgeleitet werden können.

Zuletzt werden in diesem Kapitel Anforderungen an das Framework formuliert. Die Anforderungen werden aus der Problemstellung, den verwandten Arbeiten, sowie den Anwendungsszenarien herausgearbeitet.

3.1 Problemstellung

Der Ausgangspunkt dieser Arbeit ist die in [37] beschriebene Problematik, dass REST Schnittstellen von Workflow Engines Wissen zum Umgang mit Prozessen in den Engines bei ihren Nutzern voraussetzen und nicht die Fachlichkeit der ausgeführten Prozesse, zum Beispiel durch die Verwendung fachlicher URIs und Ressourcen, abbilden. Daher werden häufig fachliche Schnittstellen entwickelt, welche die Domäne der Anwendung abbilden und die Workflow Engine verbergen. Dies ist jedoch eine zeitaufwändige und dadurch mit Kosten verbundene Tätigkeit.

Der in dieser Arbeit verfolgte Lösungsansatz führt den Gedanken von Modellausführung fort. Durch den Einsatz einer Workflow Engine muss die Prozesslogik einer Anwendung nicht mehr in Code geschrieben werden, sondern kann direkt aus einem Modell ausgeführt werden. Selbiges soll mit dem zu entwickelnden Framework auch für die fachliche Schnittstelle der Anwendung gelten. In diesem Abschnitt werden die sich daraus ergebenden Herausforderungen genauer beschrieben.

3.1.1 Modellierung komplexer Daten mit BPMN

Verschiedene Aktivitäten und Ereignisse eines Prozesses erzeugen oder verändern verschiedene Arten von Daten. Diese Daten sollen durch die generierte REST-Schnittstelle als Ressource verfügbar gemacht werden. Damit die einzelnen Ressourcen automatisch generiert werden können, wird ein Datenmodell für die Beschreibung der Ressourcen benötigt, welches mit dem Prozessmodell verbunden werden kann.

REST Ressourcen sind komplexe Datenobjekte, die sich aus mehreren, teilweise verschachtelten Attributen zusammensetzen. Darüber hinaus können Ressourcen in Relation zueinander stehen. BPMN bietet mit Datenobjekten und Datenspeichern die Möglichkeit Daten darzustellen. Allerdings lassen sich damit weder die Attribute der Daten noch ihre Beziehungen aufzeigen. Daher wird die Schnittstelle in einem separaten Modell beschrieben werden müssen.

3.1.2 Modellierung von REST-Schnittstellen

Für die Modellierung von REST-Schnittstellen existieren keine standardisierten grafischen Modellierungssprachen wie es für Geschäftsprozesse der Fall ist. Die openapi Spezifikation[1] bietet zwar einen Standard zur Dokumentation von REST-Schnittstellen, allerdings findet hierbei eine Beschreibung der Schnittstelle im JSON Format mit technischen Details statt, was für Personen ohne technischen Hintergrund schwerer verständlich ist. Die BPMN bietet grafische Modellierung von Geschäftsprozessen ohne technische Einzelheiten, damit der Prozess für alle Beteiligten verständlich ist. Um diese Zugänglichkeit weiterzuführen, muss die Modellierung der Prozessdaten, also die Modellierung der REST Ressourcen, ebenfalls in einer leicht verständlichen grafischen Modellsprache stattfinden. Dafür wird ein eigenes Modellierungswerkzeug benötigt, welches die Modelle

[1] https://www.openapis.org/

in ein standardisiertes, maschinenlesbares Format exportieren kann, so wie es auch mit BPMN Modellierungstools möglich ist.

3.1.3 Verknüpfung der einzelnen Modelle

Wenn die Prozess- und Datenbeschreibungen in unterschiedlichen Dokumenten festgehalten werden, muss eine Verknüpfung der Modelle stattfinden. Dafür ist ein Konzept nötig, mit dem sich Komponenten aus den verschiedenen Modellen gegenseitig referenzieren lassen. Hierfür muss definiert werden, wie sich die Verknüpfung der Modelle auf das Laufzeitverhalten der generierten Schnittstelle auswirkt.

3.1.4 Prozesse in Clientanwendungen

Die Verwendung von Workflow Engines ermöglicht die Trennung von Prozess- und Anwendungslogik bei der Softwareentwicklung. Dadurch können Geschäftsprozesse verändert werden, ohne dass die Anwendung, welche diese umsetzt, angepasst werden muss. Dies funktioniert zumindest dann, wenn die Engine selbständig den Anwendungscode ausführen kann. In verteilten Anwendungsszenarien ist dies nicht immer möglich. Das hat zur Folge, dass die Nachrichtenreihenfolge, mit der der Client kommuniziert, sowie Bedingungen für die Kommunikation im Clientcode festgehalten wird. Wenn Änderungen am Geschäftsprozess stattfinden, müssen dadurch auch alle Clients angepasst werden.

3.1.5 Verschiedene Produkte

Das zu entwickelnde Framework verbindet zwei Domänen, für die es bereits eine Vielzahl an Produkten gibt. Wie in 2.1.2 gezeigt, gibt es Ähnlichkeiten zwischen den verschiedenen Workflow Engines. Dennoch gibt es Unterschiede, welche die Entwicklung eines universellen Frameworks erschweren. Auch für die Implementierung von Webanwendungen existieren verschiedene Frameworks, die auf unterschiedlichen Bibliotheken basieren. Ein Framework, welches vorschreibt, welche Engine und welches Web Framework verwendet werden muss, ist für viele Anwendungsfälle nicht geeignet.

3.2 Verwandte Arbeiten

Die Problemstellung dieser Arbeit umfasst zwei Themengebiete mit eigentlich wenig Berührungspunkten. Daher findet in diesem Abschnitt zunächst eine getrennte Betrachtung von Arbeiten in den einzelnen Feldern statt. Anschließend werden Arbeiten und Konzepte vorgestellt, mit denen die einzelnen Felder näher zusammengeführt werden.

3.2.1 Modellierung und Generierung von REST-Schnittstellen

Während es für die Modellierung von Geschäftsprozessen diverse Modellierungssprachen gibt, existiert keine dedizierte Sprache, um REST-Schnittstellen zu modellieren. Dennoch existieren diverse Ansätze zu modellgetriebener Entwicklung von REST-Schnittstellen. Diese verwenden häufig ähnliche Methoden, um Eigenschaften der Schnittstellen festzuhalten. Die wichtigsten Eigenschaften sind dabei die Attribute, aus denen Ressourcen gebildet werden, ihre Zustände und die darauf anwendbaren Methoden sowie die Beziehungen zwischen den Ressourcen.

Für die Modellierung der REST Ressourcen und ihrer Beziehungen werden in verschiedenen Arbeiten UML Klassendiagramme verwendet. Dabei unterscheiden sich die Ansätze in den Vorgaben und Einschränkungen, die für die Modelle gelten. So wird in [15] ein Domänenmodell verwendet, aus dem feinere Ressourcenmodelle automatisch abgeleitet werden. Im Gegensatz dazu wird in [10, 43] ein feines Modell der einzelnen Ressourcen verwendet. Auch wenn sich in diesen Arbeiten die Vorgaben und Einschränkungen an die Modelle unterscheiden, teilen sie die Gemeinsamkeit, dass UML Klassendiagramme zur Modellierung der Daten verwendet werden. Dieser Diagrammtyp eignet sich, da REST Ressourcen ähnliche Eigenschaften haben wie Klassen aus der objektorientierten Entwicklung. So können die Attribute einer Klasse verwendet werden, um die Attribute der Ressource zu beschreiben. Um die Verbindungen zwischen einzelnen Ressourcen zu modellieren, können Assoziationen sowohl gerichtet als auch ungerichtet verwendet werden.

Für die Modellierung der Ressourcenzustände sind Zustandsautomaten ein häufig eingesetztes Mittel. Allerdings wird in einigen Arbeiten auch auf die Modellierung von Zuständen verzichtet. So werden in [43] beispielsweise CRUD Schnittstellen generiert, bei denen Ressourcen unabhängig von ihrem Zustand manipuliert werden können. Diese Vorgehensweise ist allerdings nicht für jede Anwendung geeignet, vor allem dann nicht, wenn

eine Ressource einen Lebenszyklus hat, in dem bestimmte Änderungen nur in bestimmten Zuständen möglich sind. Dann können wie in [24] Zustandsautomaten verwendet werden, um darzustellen, in welchem Zustand welche Manipulationen an einer Ressource zulässig sind.

3.2.2 Objektzentrierte Prozesse

Wie in 2.1.2 bereits beschrieben, liegt der Fokus der BPMN auf der Darstellung von Aktivitäten und weniger auf der Modellierung von Daten. Dennoch spielen Daten in Prozessen häufig eine wichtige Rolle. Es werden Daten durch Aktivitäten erzeugt oder transformiert und können für die Steuerung des Prozessablaufs verwendet werden. Objektzentrierte Prozesse arbeiten mit Daten, welche zu bestimmten Objekttypen gehören. Ein Objekttyp wird durch festgelegte Attribute dargestellt. Zur Prozesslaufzeit können von einem Objekttyp mehrere Objektinstanzen erzeugt werden. Diese können sich auch durch ihre Attribute gegenseitig referenzieren [22]. Auch wenn die BPMN Elemente zur Modellierung von Daten zur Verfügung stellt, lassen sich diese dadurch nicht detailliert beschreiben. Eigenschaften, wie die Struktur der Objekte oder Aussagen über Relationen und mögliche Werte, lassen sich nicht im Prozessmodel festhalten. Daher werden Prozessanwendungsdaten meist mit separaten Modellen und Modellsprachen festgehalten. Dieser Bruch zwischen Prozess- und Datensicht führt dazu, dass Wechselwirkungen zwischen Prozess- und Objektinstanzen nicht direkt dokumentiert werden können [40].

Darüber hinaus bieten typische Workflow Engines auch wenig Funktionalitäten, welche die Arbeit mit komplexen zusammenhängenden Objekten unterstützen. Für die Arbeit mit objektzentrierten Prozessen ist eine Modellierung der Objekte unerlässlich. WfMS bieten meist eine Anwendung zur Modellierung der Prozesse, die sich auf die Elemente der jeweiligen Modellierungssprache beschränkt. Daher können Objekte nicht ausreichend genau beschrieben werden. Mit den einfachen Mitteln wie Prozessvariablen werden lediglich die für den Prozessablauf relevanten Daten von der Workflow Engine verwaltet. Das führt dazu, dass Daten, die zum Beispiel von Bearbeitern von *Human Tasks* benötigt werden, nicht direkt über das WfMS gelesen werden können. Dadurch können umständliche manuelle Arbeitsabläufe entstehen oder weitere Systeme an Endnutzeranwendungen beteiligt werden, was die Entwicklungskomplexität erhöht [21].

Um WfMS zu entwickeln, welche eine bessere Integration von Objekt- und Prozessmodellen schaffen, werden in [23] Eigenschaften objektzentrierter Prozesse vorgestellt. Eine

für diese Arbeit wichtige Eigenschaft ist das Verhältnis zwischen Objekttypen und Prozessdefinitionen.

Wie bereits erwähnt, können sich Objektinstanzen gegenseitig referenzieren. In UML Klassendiagrammen würden für die Modellierung solcher Beziehungen Assoziationen verwendet werden. In objektzentrierten Prozessen gibt es nicht nur Beziehungen zwischen einzelnen Objekten, sondern auch zwischen Objekten und Prozessen. So ist die Erzeugung einer Prozessinstanz an die Erzeugung einer Objektinstanz gebunden. Während Aktivitäten eines Prozesses ausgeführt werden, können auch weitere Objektinstanzen erzeugt werden. Dies bedeutet, dass sich diese Beziehungen auch mit Kardinalitäten beschreiben lassen. So kann es möglich sein, dass durch eine Schleife im Prozess mehrere Objektinstanzen vom gleichen Typ erstellt werden können. Dies wäre eine 1 zu N Beziehung. Wenn der Prozess lediglich linear verläuft, besteht zwischen Prozess und Objektinstanz eine 1 zu 1 Beziehung.

Wenn Prozessinstanzen auch weitere Prozessinstanzen erzeugen können, gibt es auch Beziehungen zwischen den einzelnen Prozessmodellen, da sich diese gegenseitig referenzieren müssen. Dadurch dass die Erzeugung von Objektinstanzen an die Erzeugung der Prozessinstanzen gekoppelt ist, lassen sich aus Prozessmodellen auch Rückschlüsse auf die Assoziationen der Datenmodelle ziehen. Abbildung 3.1 verdeutlicht dies an einem Beispielbewerbungsprozess.

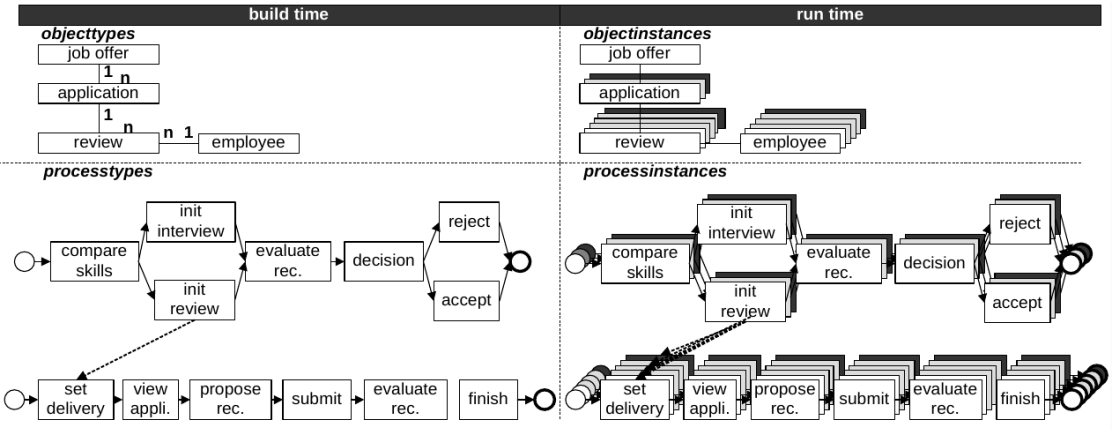


Abbildung 3.1: Beispiel für die Relation zwischen Prozessen und Daten [21]

Für eine ausgeschriebene Stelle kann es mehrere Bewerbungen geben, die mehrere Bewertungen haben. Dies wird im oberen linken Viertel der Abbildung dargestellt. Dieser

Zusammenhang, der im Datenmodell zu erkennen ist, zeigt sich auch im Prozessmodell durch die Verbindung zwischen Bewerbungprozess und Bewertungsprozess.

Diese Beziehung zwischen Prozessen und Daten wird ebenfalls zur Prozesslaufzeit, die auf der rechten Seite der Abbildung dargestellt ist, sichtbar. Jede Prozessinstanz bearbeitet ein Bewerbungsobjekt. Die dazugehörigen Bewertungen werden in den aufgerufenen Bewertungsprozessinstanzen erzeugt.

Weitere Eigenschaften von objektzentrierten Prozessen sind, dass Prozessdaten, also Objekte, jederzeit unabhängig vom Zustand des Prozesses gelesen werden können. Außerdem müssen Aktivitäten nicht zwangsweise neue Objektinstanzen erzeugen. Aktivitäten können auch die Modifikation von Attribute einer Objektinstanz darstellen.

3.2.3 Umsetzung von Prozessen mit REST-Schnittstellen

Nachdem die Rolle von Objekten in Prozessen dargestellt wurde, wird im Folgenden gezeigt, wie Prozesse durch Objekte, genauer durch REST Ressourcen, abgebildet werden können.

Der von Pautasso et al.[36] beschriebene Ansatz erweitert die BPMN durch ein neues Symbol, welches mit Prozessen und Aktivitäten verknüpft werden kann. Hierbei soll jedes mit dem Symbol verknüpfte Element als REST Ressource verfügbar gemacht werden. Für die Generierung der URIs wird die Hierarchie der Prozesse und der Aktivitäten abgebildet. Die Ressourcen repräsentieren in diesem Ansatz allerdings keine Prozessdaten, sondern die Aktivitäten und Prozesse. Es findet also keine Entkopplung von Prozess und Fachlichkeit statt.

Eine ähnliche Lösung wird in [33] vorgestellt. Das Ziel dieser Arbeit ist, Prozessmodelle dynamisch, mit Hilfe von HATEOAS, an Clients zu vermitteln. Auch hier werden keine fachlichen Daten als Ressourcen abgebildet, sondern Elemente wie Aktivitäten oder Prozesse, welche durch die Schnittstelle gesteuert werden können.

Diese Ansätze werden bereits von vielen Workflow Engines umgesetzt. So werden Prozessinstanzen oder Aktivitäten als REST Ressource verfügbar gemacht. Dies führt allerdings zu den in der Einleitung beschriebenen Nachteilen. Der Geschäftsprozess muss auch auf Clientseite implementiert werden, und Entwickler müssen sich mit den Prozessmodellen und Notationen vertraut machen.

Ein anderer Ansatz, bei dem die Prozesse und Aktivitäten nicht dem Client bekannt sein müssen, ist die Verwendung von HATEOAS mit domänenspezifischen REST-Ressourcen. Geschäftsprozesse bestimmen, in welcher Reihenfolge welche Nachrichten zwischen Server und Client ausgetauscht werden. Dadurch wird ein domänenspezifisches Anwendungsprotokoll definiert. Dieses Protokoll ist abhängig vom Zustand der Anwendung, welcher auf einem Server festgehalten wird. Damit ein Client protokollkonform mit dem Server kommunizieren kann, muss dieser zuerst den Anwendungszustand ermitteln, der durch Ressourcen repräsentiert wird. Also muss der Client Ressourcen anfragen und die Werte bestimmter Attribute prüfen. Dadurch wird Geschäftslogik auf der Clientseite implementiert. Um dies zu verhindern, können die Ressourcenrepräsentationen bereits mit Links zu möglichen Aufrufen versehen werden, was die Umsetzung von HATEOAS bedeutet. Dadurch müssen Clients nicht mehr die Werte bestimmter Attribute lesen, um die möglichen Aufrufe selber zu ermitteln. Stattdessen muss lediglich überprüft werden, ob ein bestimmter Link zu einer Methode in der Serverantwort vorhanden ist [26, 17].

Folgendes Beispiel soll dies verdeutlichen. Abbildung 3.2 zeigt einen fiktiven Lieferprozess eines Versandhandels. Hierbei wird der Prozess mit dem Empfang einer neuen Bestellung gestartet. Anschließend wird die Bestellung vorbereitet und verschickt. Der Prozess endet damit, dass die Bestellung zugestellt wurde. Dieser Prozess kann, während die Bestellung vorbereitet wird, durch eine Stornierung unterbrochen werden.

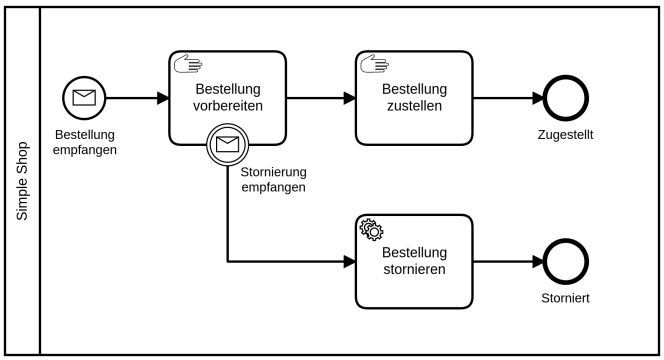


Abbildung 3.2: Standard Lieferprozess

Ohne die Verwendung von HATEOAS würde die Implementierung einer Bestellübersicht für Clients folgendermaßen aussehen. Um eine Bestellung anzuzeigen, würde die Clientanwendung eine GET-Anfrage für die entsprechende Bestellung an das Backend machen und eine Darstellung dieser Ressource als Antwort erhalten. Diese enthält eine Liste der

bestellten Waren, den Gesamtpreis, sowie den Status der Bestellung. Der Status zeigt ob die Bestellung noch vorbereitet (Listing 3.1) oder bereits geliefert wird (Listing 3.2).

Listing 3.1: Bestellung in Vorbereitung ohne HATEOAS

```
{
        "items":[{...}],
        "price":123.0,
        "status:"preparing"
}
```

Listing 3.2: Bestellung in Auslieferung ohne HATEOAS

```
{
        "items":[{...}],
        "price":123.0,
        "status":"delivering"
}
```

Um festzustellen, ob der Client den Stornierungsbutton anzeigen soll, muss dieser nun prüfen, ob die Bestellung vorbereitet wird, indem der Wert des ”status” Feldes überprüft wird. Wenn der Wert ”preparing” ist, kann der Client die Option anbieten, die Bestellung zu stornieren.

Durch diese Vorgehensweise wird Geschäftslogik im Client implementiert. Dies führt dazu, dass alle Clients angepasst werden müssen, wenn sich die Geschäftslogik ändert.

Angenommen der Versandhändler möchte zulassen, wie in Abbildung 3.3 dargestellt, dass Bestellungen, die schon geliefert werden, ebenfalls stornierbar sind. In diesem Fall müsste in allen Clients die Überprüfung des ”status” Feldes aktualisiert werden.

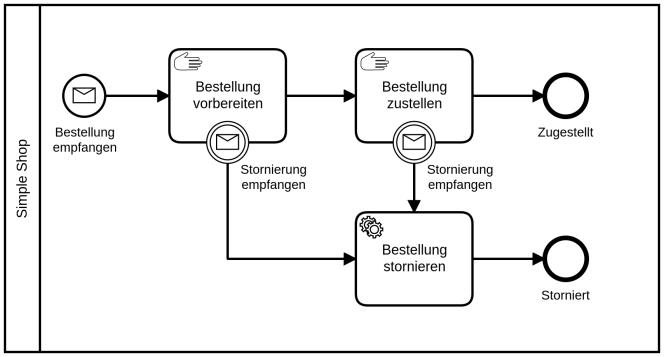


Abbildung 3.3: Lieferprozess mit kundenfreundlicher Stornierung

Durch die Verwendung von HATEOAS muss keine Überprüfung von Anwendungsdaten mehr stattfinden. Stattdessen überprüft der Client, welche Links im ”_links” Feld enthalten sind. Wenn der Geschäftsprozess wie in Abbildung 3.2 definiert ist und die Bestellung vorbereitet wird, also eine Stornierung möglich ist, wird dies durch die Anwesenheit des

”cancel” Feldes angezeigt (Listing 3.3). Wenn die Bestellung bereits versendet wird, zeigt die Abwesenheit des Feldes, dass keine Stornierung mehr möglich ist (Listing 3.4)

Listing 3.3: Bestellung in Vorbereitung mit HATEOAS

```
{
  "items": [{...}],
  "price": 123.0,
  "status":"preparing",
  "_links":
  {
  "self":"/order/123",
  "cancel":"/order/123/cancelation"
  }
}
```

Listing 3.4: Bestellung in Auslieferung mit HATEOAS

```
{
  "items":[{...}],
  "price": 123.0,
  "status":"delivering",
  {
  "self":"/order/123",
  }
}
```

Wenn der Geschäftsprozess nun verändert wird, enthält die Repräsentation einer Bestellung mit dem Status ”delivering” nun ebenfalls den Link zur Stornierung (Listing 3.5). Da die Clients bereits auf die An- beziehungsweise Abwesenheit des Links prüfen, muss keine Änderung an den Clients mehr stattfinden, um die Stornierungsoption richtig anzuzeigen.

Listing 3.5: Stornierbare Bestellung in Auslieferung mit HATEOAS

```
{
        "items":[{...}],
        "price":123.0,
        "status":"delivering",
        {
                "self" :"/order/123",
                "cancel":"/order/123/cancelation"
        }
}
```

3.3 Anwendungsszenarien

Workflow Engines sind meist in Anwendungen integriert, die komplexe änderbare Prozesse abbilden. Dabei dienen sie häufig als Middleware für Dienstkomposition in verteilten Anwendungslandschaften. In diesen Anwendungslandschaften ist die Kommunikation ein entscheidender Faktor für die erfolgreiche Ausführung eines Prozesses mit allen beteiligten Anwendungen [13]. Dieser Abschnitt stellt Szenarien vor, in denen die generierte REST-Schnittstelle eingesetzt werden kann und wie dies auf BPMN-Elemente abbildbar ist.

3.3.1 Frontend

Moderne Frontend Frameworks erstellen die Sichten, die ein Nutzer zu Gesicht bekommt, nicht mehr im Backend, sondern erstellen diese dynamisch auf dem Endgerät des Nutzers. Dafür sendet der Server dem Client bei der ersten Anfrage lediglich Code, der sämtliche Anwendungslogik für das Frontend enthält. Anschließend werden die benötigten Daten über weitere Aufrufe an eine REST-Schnittstelle abgerufen, um diese darstellen zu können. Nutzereingaben, wie ein ausgefülltes Formular, werden ebenfalls über die Schnittstelle an das Backend mitgeteilt. Auch wenn viele WfMS bereits über eine Aufgabenübersicht verfügen, wird diese nicht immer genutzt. So kann es sein, dass die Übersicht in ein bestehendes System integriert werden soll, oder dass Endgeräte verwendet werden, für die die Standardaufgabenübersicht ungeeignet ist. In diesem Fall wird meist eine fachliche Schnittstelle entwickelt, welche die Engine kapselt. Dies ist vor allem dann der Fall, wenn das Frontend uneingeschränkt im Internet verfügbar ist. Zum Beispiel für einen Onlineshop. Hier soll die fachliche Schnittstelle nicht nur die Engine verbergen, sondern auch Nutzereingaben validieren, um fehlerhafte Eingaben frühzeitig zu erkennen.

Eingaben aus einem Frontend werden in BPMN mit User Tasks dargestellt. Diese symbolisieren Aufgaben, bei denen ein Nutzer mit Hilfe von Software, meist einer Weboberfläche, Aufgaben aus einer Aufgabenübersicht bearbeitet.

Nutzereingaben können auch als Nachrichten an andere Prozessteilnehmer interpretiert werden. Dies ist vor allem dann sinnvoll, wenn der Nutzer, der die Eingaben macht, nicht Teil der Organisation ist, in der der Prozess ausgeführt wird. In diesem Fall kann

das Empfangen der Eingabe durch ein Nachrichtenereignis oder eine Empfangsaufgabe dargestellt werden.

3.3.2 Verteilte Anwendungen

Für moderne Systeme ist es nicht unüblich, dass sich diese aus einzelnen kommunizierenden Anwendungen zusammensetzen, in denen jede Anwendung eine spezielle Aufgabe übernimmt oder eine fachliche Domäne der Anwendung kontrolliert. Ein prominentes Beispiel dafür ist der Microservice Architekturstil, der schnelle Entwicklungszyklen und einfache Skalierbarkeit der Anwendungen fördern soll. Auch hier ist REST ein häufig eingesetzter Architekturstil für die Kommunikation [29].

Dabei werden die Schnittstellen nicht für Benutzereingaben verwendet, weshalb die Verwendung von User Tasks in diesem Szenario eher ungeeignet ist. Anwendungen, die an dem Prozess beteiligt sind und einzelne Aufgaben übernehmen, lassen sich durch Service Tasks darstellen. Diese symbolisieren Web Services oder andere Arten von automatisierten Anwendungen, die eine Aufgabe übernehmen. Hierbei würde eine REST Ressource das Ergebnis des Service Tasks darstellen. Alternativ dazu kann die Kommunikation der Services durch ein Nachrichtenereignis dargestellt werden.

3.3.3 API als Produkt

Ein Mischform beider Szenarien sind Schnittstellen, die für dritte Dienste angeboten werden. Diese Schnittstellen können ein eigenes Produkt sein oder als Ergänzung zu einem Produkt angeboten werden. Hierbei lässt sich nicht festlegen, ob die Daten durch einen Nutzer eingetragen oder von einem anderen Webservice generiert wurden. Allerdings handelt es sich bei solchen Schnittstellen auf jeden Fall um Daten von anderen Prozessteilnehmern, in BPMN Modellen dargestellt durch Pools, weshalb die Aktivitäten, in denen die Daten geliefert werden, auf jeden Fall durch ein Nachrichtenereignis oder eine Empfangsaufgabe dargestellt werden müssen [28].

3.4 Anforderungen

Nachdem die Problemstellung näher untersucht wurde, fasst der folgende Abschnitt die Anforderungen an ein Framework zur modellbasierten Generierung von REST-Schnittstellen für Geschäftsprozesse zusammen.

A1) Das Framework soll Aufgaben und Ereignisse eines Prozesses als REST Ressourcen anbieten.
Innerhalb eines Prozessmodells können verschiedene Aktivitäten ausgeführt und Ereignisse stattfinden, bei denen Prozessbeteiligte mit der Workflow Engine interagieren. Mit dem Framework soll die Interaktion über die generierten REST Schnittstellen stattfinden. Wie in den Anwendungsfällen beschrieben, sind Aktivitäten und Ereignisse, bei denen mit der Engine kommuniziert wird:

- Fangende Nachrichtenereignisse
- User Tasks
- Empfangsaufgaben
- Service Tasks

A2) REST Ressourcen sollen in einem UML Klassendiagramm modelliert werden.
Klassendiagramme eigenen sich für die Modellierung der Ressourcen, da sich sowohl die Attribute als auch die Beziehungen zwischen den Ressourcen darstellen lassen. Darüber hinaus sind diese auch für Personen ohne technischen Hintergrund verständlich, was den Austausch über die Prozessdaten erleichtert.

A3) Für die Beschreibung der Prozesse soll das Framework BPMN Modelle verwenden.
Die BPMN ist ein weit verbreiteter Standard, der von vielen Produkten unterstützt wird. Darüber hinaus werden BPMN Modelle in einem offenen Datenformat gespeichert, was es dem Nutzer ermöglicht, seine eigene Toolchain zu wählen.

A4) Das Ressourcenmodell soll im XMI Format eingelesen werden.
Das XMI Format ist ein offener Standard für die Beschreibung von UML Modellen, welche von verschiedenen Modellierungswerkzeugen unterstützt wird. Die Verwendung eines offenen Formats für Ressourcenmodelle knüpft an die Eigenschaft der

BPMN an und ermöglicht dem Nutzer, auch für die Modellierung der Schnittstellen, die Modellierungswerkzeuge selbst zu wählen.

A5) Das Framework soll REST-Schnittstellen mit HATEOAS unterstützen.
Die Verwendung von HATEOAS ermöglicht es, Prozesslogik von Clientanwendungen zu trennen. Dadurch können diese dynamisch auf Veränderung der Schnittstellen und der Prozesse reagieren.

A6) Das Framework soll Aktualisierung der Modelle unterstützen.
Durch die Verwendung von Workflow Engines können Prozesse schnell an neue Bedingungen angepasst werden. Dies ist möglich, da diese verschiedene Mechanismen anbieten, mit denen Prozessmodelle aktualisiert werden können. Damit kein Bruch zwischen Workflow Engine und Framework entsteht, muss dieses ebenfalls aktualisierte Modelle unterstützen.

A7) Die Verknüpfung von Prozess- und Ressourcenmodell soll über eine grafische Oberfläche stattfinden.
Durch die Verwendung grafischer Modelle lassen sich Zusammenhänge sowohl im Prozess als auch im Ressourcenmodell einfach darstellen und begreifen. Damit die Zusammenhänge zwischen den einzelnen Modellen ebenfalls leicht darzustellen und verständlich sind, sollen diese durch eine grafische Oberfläche modelliert werden.

A8) Das Framework soll die Verwendung verschiedener Workflow Engines und HTTP-Bibliotheken unterstützen.
Das Framework unterstützt zwei fachliche Domänen, für die es eine Vielzahl von Lösungen gibt. REST-Schnittstellen lassen sich mit verschiedenen Bibliotheken umsetzen. Ebenso existeren verschiedene Workflow Engines für die Ausführung von Prozessmodellen. Wenn das Framework in beiden Bereichen Vorgaben über die zu verwendenden Bibliotheken macht, unterstützt dies nur sehr wenige Anwendungsfälle. Daher wird eine Architektur benötigt, mit der die Workflow Engine sowie die HTTP-Bibliothek einfach ausgetauscht werden können.

A9) Die Verknüpfung von Prozessen und Ressourcen soll nach dem Schema objekzentrierter Prozesse stattfinden.
Die in [21] beschriebenen Eigenschaften, die für objektzentrierte Prozesse gelten, sind auch auf REST Ressourcen in Geschäftsprozessen anwendbar. Diese Eigenschaften sollen für die Verknüpfung von Daten und Prozessen während der Modellierung sowie während der Prozessausführung verwendet werden.

4 Umsetzung

Nachdem die Anforderungen an das Framework formuliert wurden, wird in diesem Kapitel die Umsetzung, von der Konzeption bis zur Implementierung, beschrieben. Dazu wird zunächst das auf objektzentrierten Prozessen basierende Konzept vorgestellt, mit dem REST Resourcen modelliert und mit Prozessen verknüpft werden können. Anschließend wird die Architektur des Frameworks erklärt. Zuletzt werden einige Aspekte aus der Implementierung vorgestellt.

4.1 Modellierung

Bevor das Framework selbst implementiert werden kann, muss eine Modellierungssprache für REST-Schnittstellen entwickelt werden. Der auf der Unified Modeling Language (UML) basierende Ansatz für die Modellierung der Ressourcen wird in 4.1.1 beschrieben.

Anschließend wird in 4.1.2 erläutert, wie das Ressourcenmodell mit den Prozessmodellen verknüpft werden soll. Die Verknüpfung umfasst zwei Aspekte: die Verbindung der Modelle innerhalb der Dateien sowie die Ausführungssemantik.

Zur Anwendungslaufzeit können Modelle auch verwendet werden, um empfangene Daten zu validieren. In Abschnitt 4.1.3 werden dazu einige Designentscheidungen erläutert.

Zuletzt wird das Kapitel in 4.1.4 zusammengefasst, indem das Metamodell zur erarbeiteten Lösung vorgestellt wird.

4.1.1 REST Ressourcen

Wie in der Übersicht über verwandte Arbeiten beschrieben, existieren verschiedene Ansätze zur Modellierung von REST Ressourcen. Eine Gemeinsamkeit, die sich alle vorgestellten Lösungen teilen, ist die Verwendung von UML-Klassendiagrammen, welche auch in dieser Arbeit eingesetzt werden sollen. Klassendiagramme bieten eine Vielzahl an Modellierungselementen, was die Entwicklung eines Parsers sowie das Generieren von Ressourcen und den dazugehörigen URIs kompliziert macht. Da der Fokus dieser Arbeit auf der Generierung von Schnittstellen für Geschäftsprozesse liegt, und die Generierung von REST-Schnittstellen aus vollwertigen Klassendiagrammen oder Domänenmodellen über das Ziel dieser Arbeit hinausgehen würde, werden einige Regeln für die Datenmodelle festgelegt, die im Folgenden erklärt werden. Dabei wird nur auf die Struktur und nicht auf das Verhalten der Schnittstelle eingangen. Dieser Aspekt der Schnittstelle wird in der Modellverknüpfung weiter vertieft.

Ressourcen

Die Modelle der Ressourcen müssen drei Dinge abbilden: den Namen der Ressource, die Attribute, aus denen sich eine Ressource zusammensetzt sowie die Beziehungen zwischen den Ressourcen. Im Klassendiagramm stellt eine Klasse jeweils eine Ressource dar, diese hat einen Namen und Attribute. Da die Methoden einer Ressource durch das HTTP-Protokoll vorgegeben sind, müssen diese nicht im Modell festgehalten werden. Bei der Modellierung von Attributen kann mit der UML die Sichtbarkeit dieser festgelegt werden. Das Konzept von privaten oder geschützten Attributen ist für REST-Schnittstellen nicht anwendbar, da REST Ressourcen über Systemgrenzen hinweg ausgetauscht werden. Darüber hinaus ist der Zweck der Schnittstelle, dass Kommunikation in einem verteilten System ermöglicht wird. Dabei ist es überflüssig, Attribute auszutauschen, die nicht gelesen werden sollen. Daher kann bei der Modellierung von REST Ressourcen auf die Kennzeichnung der Sichtbarkeit von Attributen verzichtet werden, da standardmäßig alle sichtbar sein sollen. Abbildung 4.1 zeigt das Modell für eine Ressource mit dem Namen *Application* und ihren Attributen. Hierbei ist die Sichtbarkeit nicht gekennzeichnet, und es wurden auch keine Operationen aufgezählt.

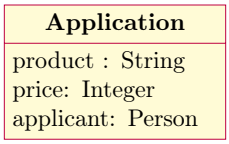


Abbildung 4.1: Beispielressource mit UML

Assoziationen

Neben den Eigenschaften der Ressourcen muss das Modell noch die Beziehungen zwischen den Ressourcen festhalten. Dafür bietet die UML eine Vielzahl an Modellierungselementen. Um die Modellierung und die Verarbeitung der Modelle einfach zu halten, müssen die Ressouren einen Baum aus Assoziationen bilden.

Für REST-Schnittstellen ist es typisch, dass die URIs, welche für die Identifikation von Ressourcen verwendet werden, eine Ressourcenhierarchie abbilden [13]. Dies ist zwar nicht zwingend notwendig, trägt aber zur Verständlichkeit der Schnittstelle bei. Diese Hierarchie bildet die Zugehörigkeit einer Kindressource zu ihrer Elternressource ab und soll im Modell durch den Baum, den die Assoziationen bilden, dargestellt werden.

Damit Ressourcen des gleichen Typs unterschieden werden können, setzen sich URIs nicht nur aus aneinandergereihten Ressourcennamen ihrer Hierarchie zusammen, sondern auch aus Identifiern, die an den benötigten Stellen eingefügt werden. Identifier müssen immer dann eingesetzt werden, wenn eine Ressource mehrere Kindressourcen des gleichen Typs haben kann. Um kennzuzeichnen, dass mehrere Kindressourcen existieren können, müssen Kardinalitäten an den Assoziationen des Ressourcenmodels verwendet werden.

Die für die Generierung der URIs wichtigste Unterscheidung der Kardinalitäten ist hierbei, ob eine Ressource eine oder mehr als eine Kindressource, des gleichen Typs haben kann. Wenn mehrere Kindressourcen existieren können, müssen diese durch einen Identifer unterschieden werden. Ressourcen, die nur einmal erzeugt werden können, können durch den Identifier der Elternressource identifiziert werden. Neben Kindressourcen, die mehr als einmal erzeugt werden können, wird für die Wurzelressource ebenfalls ein Identifier verwendet, damit auch hier eine Unterscheidung der einzelnen Instanzen möglich ist. Typischerweise liefern URIs ohne Identifier, bei Ressourcen die mehr als einmal vorkommen können, eine Sammlung der einzelnen Ressourcen. Darüber hinaus wird der URI ohne Identifier auch zur Erzeugung neuer Instanzen verwendet.

Abbildung 4.2 zeigt ein fiktives Ressourcenmodell für die Schnittstelle einer Versicherung auf Grundlage der genannten Vorgaben. Hierbei können mehrere Dokumente zu einem Antrag gehören. In jedem Fall hat ein Antrag eine Entscheidung. Diese kann eine Bestätigung haben.

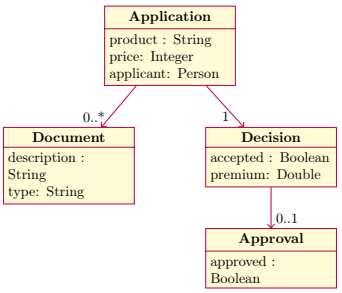


Abbildung 4.2: Ressourcenbaum aus gerichteten Assoziationen

Aus diesem Ressourcenbaum würden folgende URIs abgeleitet werden:

- /application
- /application/{applicationID}
- /application/{applicationID}/document
- /application/{applicationID}/document/{documentID}
- /application/{applicationID}/decision
- /application/{applicationID}/decision/approval

Die erste URI würde eine Liste aus Antragsressourcen zurückgeben. Außerdem lassen sich darüber neue Ressourcen erzeugen. Die zweite URI zeigt, wie einzelne Antragsressourcen adressiert werden. Da Dokumente mehr als einmal existieren können, wird für einzelne Instanzen ebenfalls ein Identifier benötigt. Die letzten zwei Pfade zeigen, wie URIs bei Ressourcen gebildet werden, die höchstens einmal oder weniger für eine Elternressource existieren. Sowohl die Entscheidungsressource, als auch die Bestätigungsressource wird über den Antrag identifiziert.

4.1.2 Modellverknüpfung

Nachdem erläutert wurde, wie das Ressourcenmodel auszusehen hat und wie daraus Ressourcen generiert werden können, wird im Folgenden gezeigt, wie das Modell mit dem Prozessmodell verknüpft wird und wie sich die Verknüpfung auf das Schnittstellenverhalten auswirkt.

Nach dem Prinzip der objektzentrierten Prozesse sind Objekte und Prozesse stark miteinander verknüpft. Ein Prozess kann ein Objekt erzeugen oder verarbeiten. Das Gleiche gilt auch für Aktivitäten oder Nachrichtenereignisse. Dieses Prinzip ist auch mit der typischen Definition von Geschäftsprozessen, wie zum Beispiel der von Davenport, vereinbar.

> In definitional terms, a process is simply a structured, measured set of activities designed to produce a specified output for a particular customer or market [8, p. 5].

Der Output eines Prozesses oder einer Aktivität lässt sich durch ein Objekt, in unserem Fall durch eine REST Ressource, darstellen. Außerdem kann die Ressource die Nachricht eines Nachrichtenereignisses darstellen. Dadurch entsteht die in [21] beschriebene Verbindung zwischen Prozess- und Objektinstanzen.

Damit das Framework die Verbindungen zwischen den verschiedenen Instanzen herstellen kann, müssen diese im Modell festgehalten werden. Dazu wurden zwei verschiedene Ansätze erarbeitet, die unterschiedliche Elemente der BPMN verwenden. Der erste Ansatz basiert auf BPMN Extension Elements, der zweite auf Datenobjekten.

Verknüpfung durch Extension Elements

Extension Elements dienen dazu, beliebige BPMN Elemente mit Daten, die nicht in der Spezifikation vorgesehen sind, zu erweitern. Um eine Ressource mit einem Prozess oder einer Aktivität zu verknüpfen, kann ein neues Element verwendet werden, mit dem die Ressource referenziert wird. Listing 4.1 zeigt einen User Task mit dem Namen Approval. In Zeile 3 ist ein Extension Element mit dem Namen ”LinkedRessource”, welches über das Attribut "name", den Namen der mit dem Task verknüpften Ressource angibt.

Listing 4.1: BPMN User task mit Extension Element

```
1 <bpmn:userTask id="Activity_04bi4lp" name="Approve">
2         <bpmn:extensionElements>
3                 <linkedResource name="approval"/>
4         </bpmn:extensionElements>
5         <bpmn:incoming>Flow_1dzcmml</bpmn:incoming>
6         <bpmn:outgoing>Flow_0wx3he6</bpmn:outgoing>
7 </bpmn:userTask>
```

Dieser simple Ansatz fügt sich in die BPMN ein, ohne die Semantik der existierenden Elemente zu beeinflussen, allerdings muss hierbei das Extension Element manuell an der richtigen Stelle in der Prozessdefinition eingefügt werden. Einige Modellierungswerkzeuge für Prozesse können erweitert werden, sodass die Verknüpfung nicht händisch in der XML Datei umgesetzt werden muss. Allerdings sind diese Erweiterungen nicht direkt im Modell durch ein grafisches Element sichtbar.

Verknüpfung durch Datenobjekte

Durch die Verwendung von Datenobjekten können verknüpfte Ressourcen sichtbar im Modell dargestellt werden. Abbildung 4.3 zeigt, wie die Verknüpfung, die zuvor durch das Extension Element realisiert wurde, mit einem Datenobjekt umgesetzt werden kann. Die ausgehende Datenassoziation zeigt an, dass der User Task *Approve* die Ressource *approval* erzeugt. Ein Nachteil bei diesem Vorgehen ist jedoch, dass nicht alle BPMN Elemente mit Datenobjekten verknüpft werden können. So ist es zum Beispiel nicht möglich, anzuzeigen, dass ein Prozess eine Ressource erzeugt, da Lanes und Pools nicht mit Datenobjekten verbunden werden können.

Verknüpfung zur Laufzeit

Nachdem gezeigt wurde, wie die Modelle miteinander verknüpft wurden, erklärt dieser Abschnitt, wie sich die Verbindungen auf das Verhalten der generierten Schnittstelle und der Prozesse auswirkt.

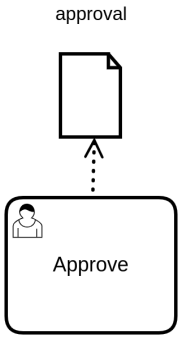


Abbildung 4.3: Verknüpfung von Ressource mit Datenobjekt

URIs Wenn eine Ressource mit einer Aktivität verknüpft wurde, wird diese durch die Aktivität erzeugt oder verändert. Das heißt, dass das Framework einen Endpunkt zum Lesen und zur Manipulation der Ressource zur Verfügung stellen muss. Wie die Endpunkte für die Ressourcen aus dem Datenmodell generiert werden, wurde bereits im Abschnitt 4.1.1 erläutert. Zur Laufzeit kommt noch eine Anforderung für die URIs hinzu. Es muss für das Framework möglich sein, mit Hilfe der URIs die zu der Ressource gehörenden Prozess- und Aktivitätsinstanzen, die in der Process Engine ausgesführt werden, zu sammeln. Diese Verbindung wird im Framework durch die in den URIs zu setzenden Identifier (IDs) realisiert, indem die IDs der Prozessinstanzen sowie Business Keys verwendet werden. Business Keys sind nicht in der BPMN spezifiziert, allerdings wird dieser Mechanismus von vielen Workflow Engines angeboten. Business Keys sind Identifier, die von einer aufrufenden an eine aufgerufene Prozessinstanz weitergegeben werden. Da alle zusammengehörenden Instanzen den selben Business Key haben, ist es für Anwendungsentwickler leichter, diese zu finden.

Für die Unterscheidung der Wurzelressourcen wird in den URIs der Business Key verwendet. Damit können alle Prozessinstanzen gefunden werden, in denen die Wurzelressource möglicherweise bearbeitet wird. Durch den Business Key können außerdem Subressourcen, die nicht mehr als einmal existieren können, gefunden werden. Für Ressourcen, die mehr als einmal erzeugt werden können, muss eine Einschränkung für die Modelle beachtet werden. Diese Subressourcen können nur in einer separaten Prozessdefinition vorkommen, die als Call Activity gestartet wird. Dies ist erforderlich, damit die Subressourcen anhand der Prozessinstanzidentifier unterschieden werden können.

Die Generierung der URIs soll anhand des Beispielprozesses aus Abbildung 4.4 und 4.5 verdeutlicht werden. Das Model beschreibt den manuellen Zustimmungsprozess einer

Versicherung sowie die dazugehörige Dokumentenanforderung. Das Ressourcenmodell der Prozesse wurde bereits in Abbildung 4.2 dargestellt. Für die Verknüpfung von Aktivitäten und Ressourcen wurden Datenobjekte verwendet. Die Lane in Abbildung 4.4 ist durch ein Extension Element mit der *Application* Ressource verknüpft.

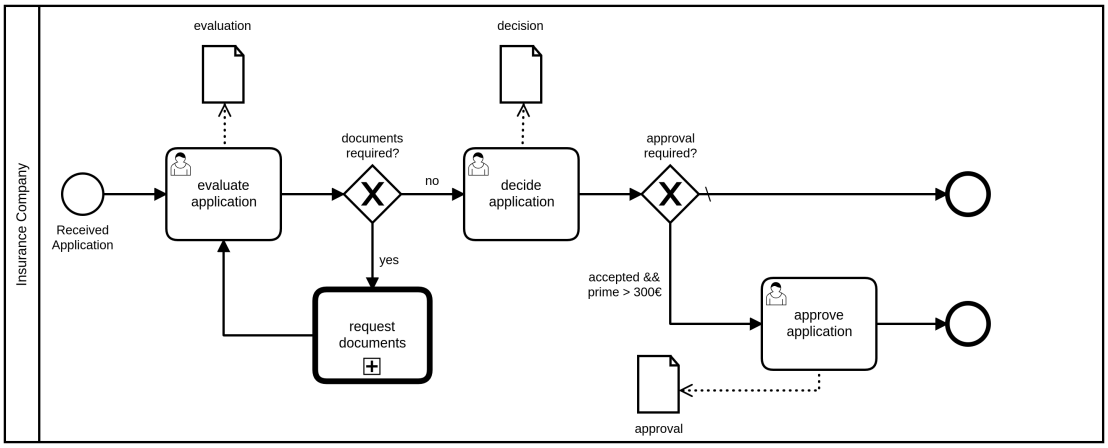


Abbildung 4.4: Prozess Antrag prüfen

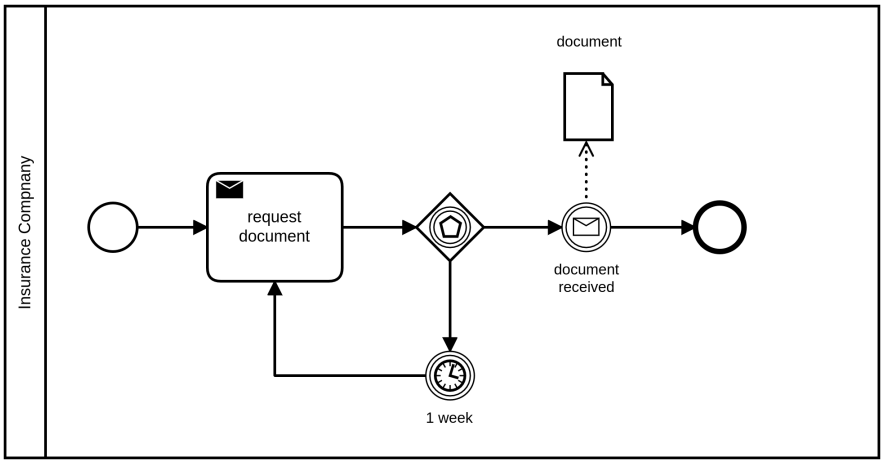


Abbildung 4.5: Prozess Dokument anfordern

Für dieses Beispiel würden die URIs folgendermaßen gebildet werden.

- /application
- /application/{businessKey}
- /application/{businessKey}/document

- /application/{businessKey}/document/{processInstanceID}
- /application/{businessKey}/decision
- /application/{businessKey}/decision/approval

Durch den Business Key können Beurteilungs-, Entscheidungs- und Bestätigungsressourcen gefunden werden. Diese kommen im gesamten Prozess höchstens einmal vor, weshalb einzelne Instanzen nicht durch weitere Identifier unterschieden werden müssen. Da es im Prozess mehrere Dokumente geben kann, wird zur Unterscheidung der einzelnen Dokumentenressourcen die ID der Prozessinstanz für die Generierung der URI verwendet.

Zustände Eine Ressource, die durch die REST-Schnittstelle erreichbar ist, soll nicht zu jeder Zeit bearbeitet werden können. Der Zustand einer Aktivität gibt vor, welche Operationen auf die verknüpfte Ressource anwendbar sind. Da die Ressource das Ergebnis einer Aktivität repräsentiert, kann sie auch erst dann bearbeitet werden, wenn die Aktivität aktiv ist. Selbiges gilt auch für Empfangsereignisse. Das Framework muss also den Zustand der Aktivität oder des Ereignisses mit dem der Ressource verbinden.

Eine Ressource muss immer dann bearbeitet werden können, wenn die Aktivität ausgeführt wird. Dies ist, wie in Abschnitt 2.1 beschrieben, im Zustand *aktiv* möglich. In den übrigen Zuständen, die eine Aktivität annimmt, kann die Ressource nicht bearbeitet werden, da hierbei die Zustandsänderungen durch die Workflow Engine gesteuert werden.

Die Information, ob eine Ressource bearbeitbar ist, kann auch für die Implementierung von HATEOAS genutzt werden.

4.1.3 Validierung von Ressourcen

Für die Validierung von Ressorcen sind, bis auf die Datentypen, keine Informationen in den Modellen vorgesehen. Dies hat zur Folge, dass die Validierung manuell implementiert werden muss. Auch wenn es mit der Object Constraint Language (OCL) [35] einen Ansatz zur Modellierung von Bedingunen in UML gibt, wird diese nicht genutzt. Die Verwendung der OCL würde die Anwendungsentwicklung mit dem Framework erschweren, da eine weitere Programmiersprache zur Anwendung hinzukommt, die erlernt werden muss. Darüber hinaus müssen Entwicklungswerkzeuge für die Sprache in Form von IDEs und

Testbibliotheken bereitgestellt werden. Abschnitt 4.3.3 zeigt, wie die Validierung in der Implementierung des Frameworks umgesetzt wird.

4.1.4 Metamodell

In diesem Kapitel wurde gezeigt, wie Ressourcen modelliert werden sollen und wie diese mit Prozessmodellen verknüpft werden. Darüber hinaus wurde erklärt, wie diese Verbindungen das Laufzeitverhalten der Ressourcen und des Prozesses bestimmen. Dieser Abschnitt fasst das Kapitel zusammen und verdeutlicht die Beziehungen zwischen den Modell- und Laufzeitelementen anhand des Metamodells aus Abbildung 4.6.

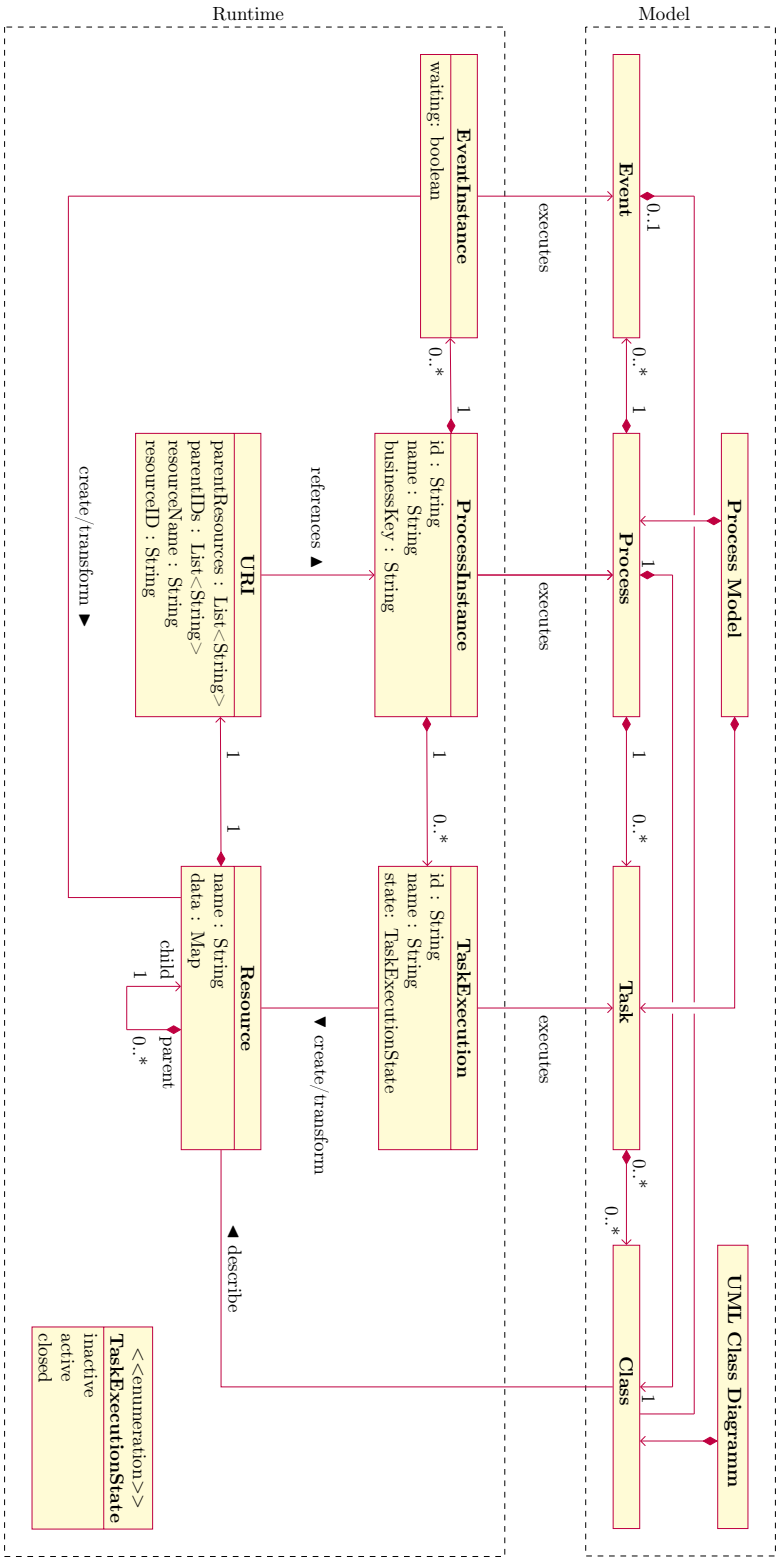


Abbildung 4.6: Metamodell

Der obere Teil des Modells zeigt den Zusammenhang zwischen den Modellen und ihren Modellierungselementen. Dabei kann ein Prozess, sowie die zugehörigen Ereignisse und Aktivitäten mit einer Klasse aus dem Klassendiagramm verknüpft werden. Das Metamodell zeigt nicht, wie Extension Elemente oder Datenobjekte für die Verknüpfung eingesetzt werden, da der verwendete Verbindungsmechanismus keinen Einfluss auf das Verhalten der Schnittstelle hat und das Modell unnötig komplex werden würde.

Eine Klasse aus dem Klassendiagramm repräsentiert jeweils eine Ressource. Diese wird durch die dazugehörige URI identifiziert und kann mehrere Subressourcen haben. Das Mapping zwischen der datenorientierten Sicht und der Prozesssicht findet durch die URI statt, indem diese zu genau einer Ressource gehört und gleichzeitig Prozessinstanzen referenziert, indem sie die Business Keys und IDs verwendet. Die zu einer Prozessinstanz zugehörigen Aktivitäts- und Ereignisausführungen werden durch die Ereignisse und Aktivitäten aus dem Prozessmodell beschrieben. Durch die Verbindung zu einer Klasse aus dem Klassendiagramm wird beschrieben, welche Ressourcen die Ausführungsinstanzen zur Laufzeit manipulieren können. Wichtig ist, dass eine Ereignisinstanz eine Ressource immer nur dann verändern kann, wenn diese auf das Eintreten des Ereignisses wartet, also wenn der Prozessverlauf an diesem Ereignis angekommen ist. Für Aktivitätsausführungen gilt, dass diese nur dann Ressourcen manipulieren können, wenn sie sich im Zustand *aktiv* befinden. Die Zustände, welche im Enum *TaskExecutionState* aufgelistet sind, sind aus dem Automaten, der in 2.1 beschrieben wurde, entnommen, wobei, um das Model einfach zu halten, nur die wichtigsten Zustände verwendet wurden.

4.2 Architektur

Dieser Abschnitt beschreibt die Architektur des Frameworks mit Hilfe der 4 Sichten nach Starke [42]. Diese bestehen aus einer Kontextabgrenzung und einer Bausteinsicht, welche die wichtigsten Komponenten des Frameworks vorstellt. Darüber hinaus wird das Laufzeitverhalten, sowie mögliche Deployment-Szenarien beschrieben.

Die Architektur soll die Kompatibilität des Frameworks mit verschiedenen Workflow Engines und HTTP-Bibliotheken fördern.

4.2.1 Kontextabgrenzung

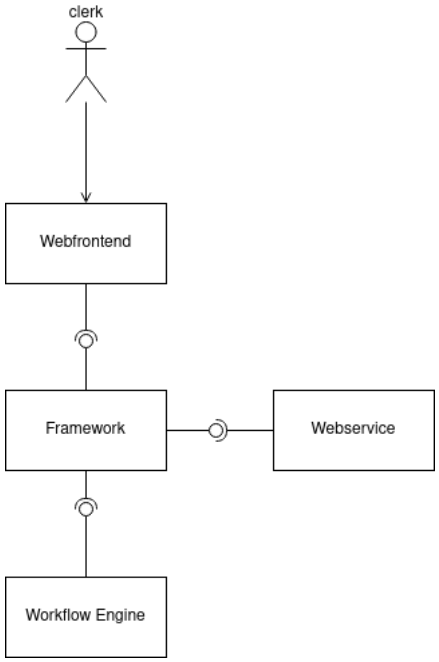


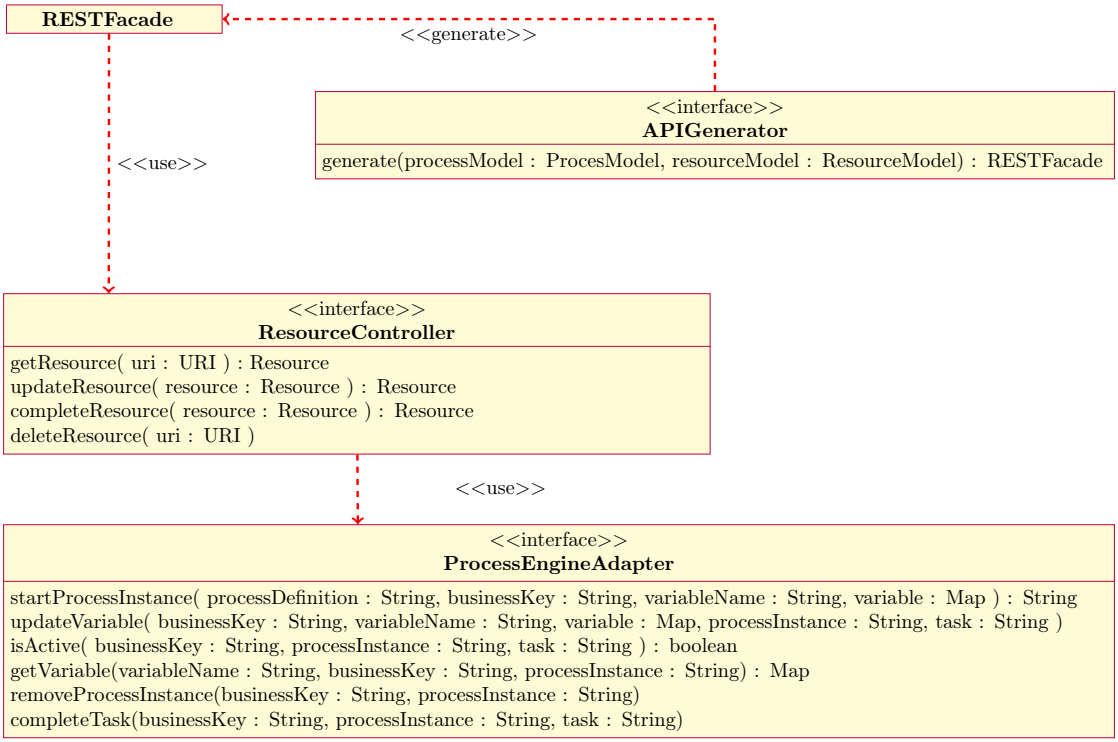
Abbildung 4.7: Kontextsicht

Die Kontextsicht, beschreibt, wie das Framework in die Umgebung eingebettet ist. Dabei sollen Abhängigkeiten von und zu anderen Services beschrieben sowie mögliche Endnutzer aufgezeigt werden. Da es sich hierbei nicht um eine konkrete Anwendung handelt, die entwickelt wird, können auch keine konkreten Systeme im Diagramm aufgezeigt werden. Stattdessen werden die bereits beschriebenen Anwendungsfälle visualisiert.

Wie in Abbildung 4.7 dargestellt verwendet das System die Schnittstelle einer Workflow Engine, um die Prozese zu steuern und ein Mapping von Datensicht auf Prozesssicht durchzuführen. Weitere Systeme können diese Schnittstelle dann nutzen. Dies können andere Webservices oder Frontends sein, die von Endnutzern bedient werden.

4.2.2 Bausteinsicht

Die Bausteinsicht soll die wichtigsten Komponenten der Anwendung festhalten, um die Abhängigkeiten sowie die benötigten Schnittstellen darzustellen.



Das Framework setzt sich aus vier Komponenten zusammen. Die Komponente *API Generator* soll aus dem Prozess- und dem Datenmodell eine REST-Facade generieren. Da die Facade eine bestimmte HTTP-Bibliothek implementiert, muss der API-Generator an die verwendete Bibliothek angepasst werden. Die zur Laufzeit generierte Facade nimmt die REST Aufrufe entgegen und ruft die *Resource Controller* Komponente auf.

Diese Komponente ist nicht von den eingesetzten Produkten abhängig. Das Interface bietet die für REST-Schnittstellen typischen CRUD Operationen an. Innerhalb dieser Komponente findet das Mapping von Daten zu Prozesssicht sowie die Verifikation der empfangenen Daten statt. Informationen über Prozessinstanzen, Aktivitäten und Ereignisse, die für das Mapping benötigt werden, erhält die Komponente über Aufrufe an den *Process Engine Adapter*.

Dieses Interface dient als Abstraktionsschicht für die verwendete Workflow Engine. Daher muss dieses Interface für jede Engine, die das Framework unterstützen soll, implementiert werden. Die angebotenen Methoden werden vom *Ressource Controller* verwendet, um Prozessvariablen zu setzen, den Status von Aktivitäten abzufragen oder Ereignisse auszulösen.

4.2.3 Laufzeitsicht

Die Laufzeitsicht beschreibt das Systemverhalten und die Interaktion der Komponenten zur Laufzeit. In diesem Abschnitt wird dazu exemplarisch gezeigt, wie eine Aktivität durch die REST API beendet wird. Abbildung 4.8 stellt dies in einem Sequenzdiagramm dar.

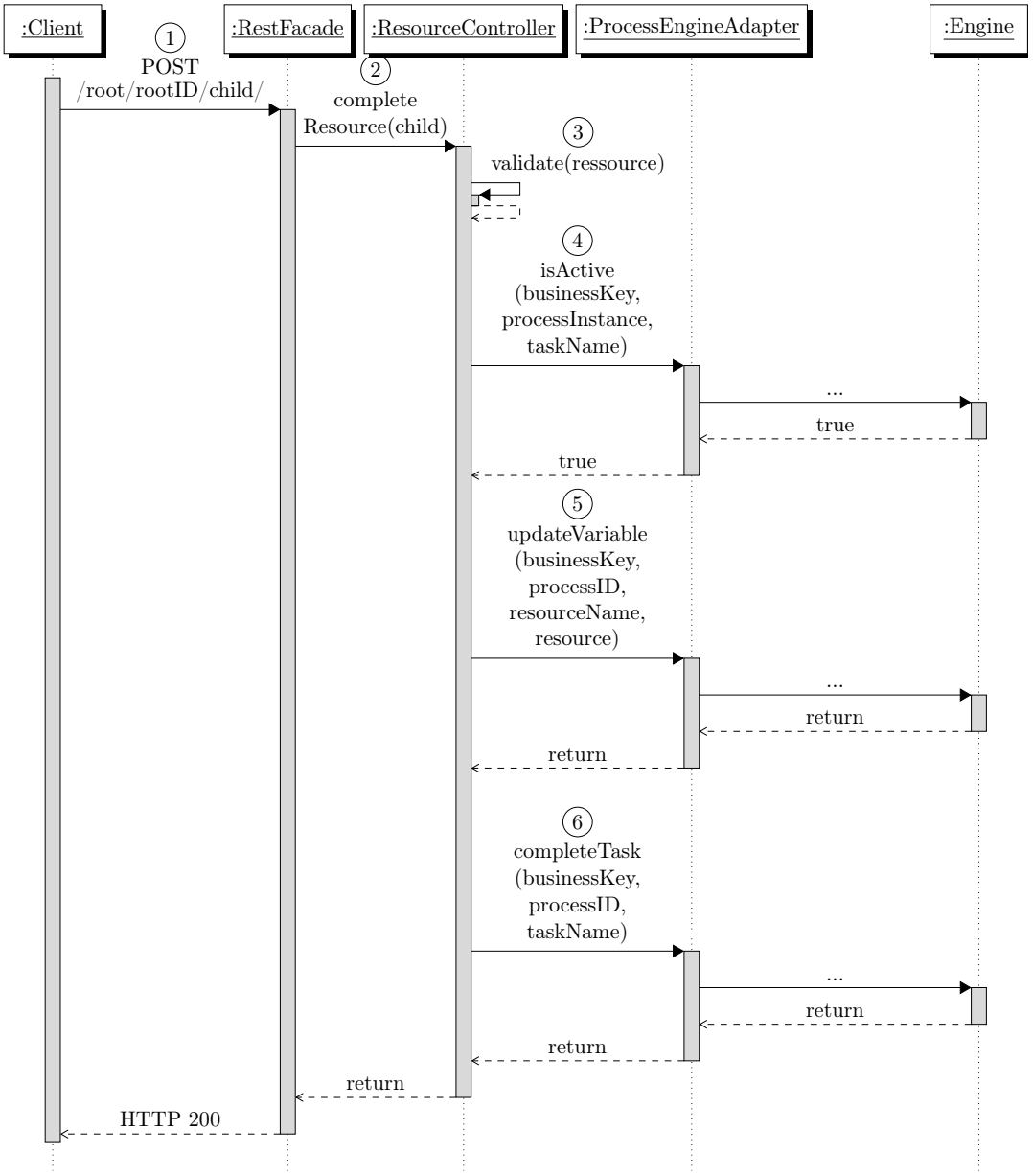


Abbildung 4.8: Laufzeit

Im Folgenden werden die nummerierten Aufrufe im Diagramm erläutert, wobei die Aufrufe an die Engine ausgelassen wurden, da sich diese bei verschiedenen Produkten unterscheiden.

1. Ein Client sendet eine HTTP POST-Anfrage an die generierte REST-Facade. Dabei signalisiert die POST-Methode, dass ein Zustandswechsel der Aktivität erwünscht ist. Derselbe Aufruf mit der PUT-Methode würde lediglich die Werte der Ressource aktualisieren, ohne dass sich der Zustand der Aktivität ändert. Dieses Verhalten trägt zur Konformität der generierten Schnittstelle mit der HTTP-Spezifikation bei. Demnach müssen PUT-Aufrufe idempotent sein [30]. Wenn sich durch einen Aufruf mit der PUT-Methode der Zustand einer Aktivtät ändert, ist dies nicht mehr sichergestellt. Daher werden POST-Aufrufe verwendet.

2. Die generierte Facade erstellt aus der HTTP-Anfrage ein Ressourcenobjekt, wie in 4.6 beschrieben, mit zugehöriger URI und ruft damit die Methode *completeResource* des *Ressource Controllers* auf.

3. Der *Resource Controller* validiert die empfangene Ressource und prüft, ob diese gültig ist.

4. Anschließend prüft der Controller, ob die Ressource überhaupt bearbeitet werden kann, indem nachgesehen wird, ob eine mit der Ressource verknüpfte Aktivität aktiv ist. Dazu macht der Controller Aufrufe an die Engine, mit dem Engine Adapter als Abstraktionsschicht.

5. Wenn eine Aktivität aktiv ist, aktualisiert der Controller die Prozessvariable, welche den Zustand der Ressource, durch einen Aufruf an den Adapter, speichert. Die für den Aufruf benötigten IDs werden aus dem URI-Objekt der Ressource entnommen.

6. Der Controller beendet die Aktivität.

4.2.4 Verteilungssicht

Die Verteilungssicht beschreibt die Ablaufumgebung, in der eine Anwendung ausgeführt wird. Dazu gehören die Server, auf denen die Anwendung ausgeführt wird, aber auch virtuelle Ausführungsumgebungen. Darüber hinaus wird die Anbindung an Nachbarsysteme und, wie damit kommuniziert wird, festgehalten.

Da das Framework mit verschiedenen Engines verwendet werden kann, können dadurch auch verschiedene Deploymentszenarien umgesetzt werden.

Das simpelste Szenario ist in Abbildung 4.9 dargestellt. Hierbei wird eine einzige Jar Datei ausgeführt, welche die Engine und das Framework beinhaltet. Die Kommunikation mit der Engine findet über die von ihr bereitgestellten Java Schnittstellen statt.

Alternativ dazu kann wie in Abbildung 4.10 dargestellt eine Trennung von Engine und Framework stattfinden. Die Abhängigkeiten müssen dann zur Laufzeit aufgelöst werden. Auch hier wird über die Java Schnittstellen kommuniziert.

Das letzte Szenario sieht die getrennte Ausführung von Engine und Framework vor. Dies setzt allerdings voraus, dass diese Engine eine REST-Schnittstelle anbietet, über die das Framework mit der Engine kommunizieren kann. Dieses Szenario wird durch die Abstraktion der Engine, die der Adapter bietet, ermöglicht. Für die Umsetzung dieses Szenarios muss die Implementierung des Adapter Interfaces die REST-Schnittstelle der Engine verwenden.

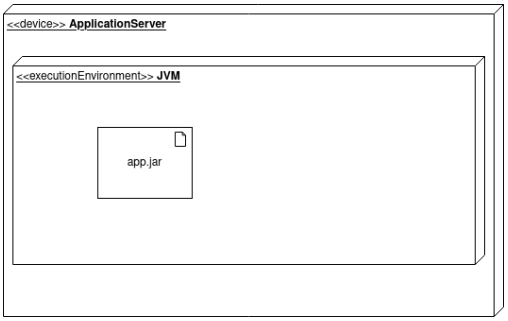


Abbildung 4.9: Deployment Szenario: Framework und Engine in einer Datei

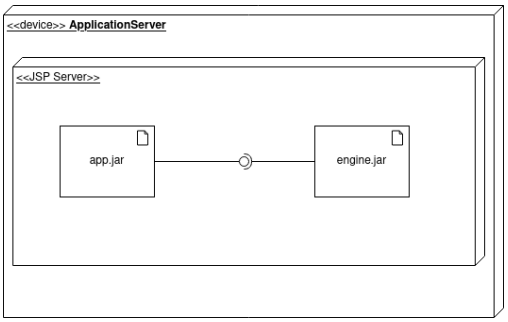


Abbildung 4.10: Deployment Szenario: Framework und Engine werden zur Laufzeit verknüpft

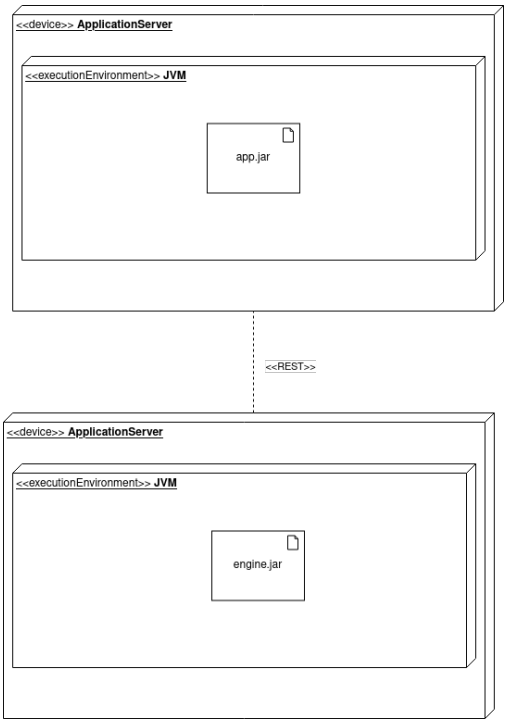


Abbildung 4.11: Deployment Szenario: Framework und Anwendung kommunizieren über REST

4.3 Implementierung

Dieser Abschnitt beschreibt Details zur Implementierung des Frameworks und gibt einen Überblick über die verwendeten Technologien.

4.3.1 Technologien

Das Framework unterstützt aktuell die Camunda Worfklow Engine im lokalen Deploymentszenario und Spring als HTTP Library. Durch weitere Implmentierungen des Adapters können weitere Engines hinzugefügt werden.

Für die Implementierung des Schnittstellengenerators wurden funktionale Router[1] verwendetet. Im Gegensatz zu annotationsbasierten Routern sind diese als Objekt verfügbar, was es ermöglicht, das Verhalten zur Laufzeit zu bestimmen.

Für die interne Repräsentation des Ressourcenmodells wurde die Graphbibliothek JGraphT[2] verwendet. Diese ermöglicht einfache Graphtraversierung für die Ermittlung von Eltern- und Kindrelationen der Ressourcen zur Laufzeit

4.3.2 Application Starter für erleichtertes Deployment

Um das Deployment der Anwendung zu vereinfachen, wurde ein Spring Starter für das Framework entwickelt. Durch die Verwendung des Starters muss das Framework nur als eine Abhängigkeit im Build Tool(Gradle/Maven) hinzugefügt werden, um eine lauffähige Anwendung zu erzeugen.

Der Starter unterstützt das Deployment des Frameworks als lauffähige Jar Datei. Für andere Deploymentszenarien muss die Konfiguration des Frameworks manuell umgesetzt werden.

4.3.3 Validierung von Ressourcen

In der Implementierung des Frameworks wird die Validierung von Ressourcen vom RessourceController an eine Implementierung des Interfaces RessourceValidator delegiert. Abbildung 4.12 zeigt den Zusammenhang zwischen der Implementierung der RessourceController Komponente und dem RessourceValidator Interface.

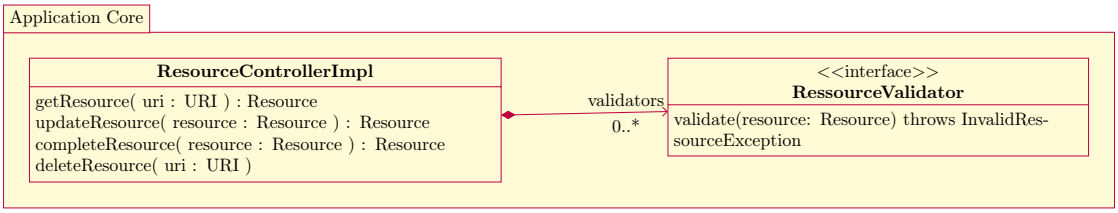


Abbildung 4.12: Validierung von Ressourcen mit dem RessourceController in der Implementierung

[1] https://spring.io/blog/2016/09/22/new-in-spring-5-functional-web-framework
[2] https://jgrapht.org/

Das Interface muss für jede Ressource manuell implementiert werden. Die Methode ”validate” der Implementierung erhält eine REST Ressource und muss eine Excpetion auslösen, falls das übergebene Objekt nicht die erforderten Bedingungen erfüllt. Dabei kann jeder Ressourcentyp durch mehrere Implementierungen überprüft werden.

Damit das Framework weiß, welche Ressource mit welcher Implementierung des Interface überpüft wird, muss eine Zuordnung von Implementierung zu Ressource stattfinden. Dies kann auf zwei Arten umgesetzt werden. Der Klasse ResourceControllerImpl kann bei der Erzeugung ein Map Objekt übergeben werden, in dem die Ressourcennamen auf die jeweiligen Implementierungen verweisen. Alternativ können, wenn der Application Starter verwendet wird, die RessourceValidator Implementierungen mit Java Annotationen versehen werden, welche auf die Ressource verweisen.

4.3.4 Stand der Umsetzung

A1) Das Framework soll Aufgaben und Ereignisse eines Prozesses als REST Ressourcen anbieten.
Das Framework unterstützt die Interkation mit

- Fangenden Nachrichtenereignissen
- User Tasks
- Empfangsaufgaben

A2) REST Ressourcen sollen in einem UML Klassendiagramm modelliert werden.
Das Framweork unterstützt Klassendiagramme, die nach den in 4.1.1 beschriebenen Vorgaben aufgebaut sind.

A3) Für die Beschreibung der Prozesse soll das Framework BPMN Modelle verwenden
Das Framework kann mit allen BPMN-konformen Modellierungstools verwendet werden.

A4) Das Ressourcenmodell soll im XMI Format eingelesen werden.
Aktuell können XMI Modelle nach der Spezifikationsversion 2.5 verwendet werden. Getestet wurde dies mit dem Modellierungstool StarUML[3]

[3] https://staruml.io/

A5) Das Framework soll REST-Schnittstellen mit HATEOAS unterstützen.
Es werden die Eltern- und Kindressourcen sowie die möglichen Aufrufe mit Hilfe der Hypertext Application Language (HAL) als Teil der Serverantwort kommuniziert.

A6) Das Framework soll Aktualisierung der Modelle unterstützen.
Die Aktualisierung der Prozessmodelle geschieht durch die Workflow Engine. Sofern Aktivitäten und Ereignisse mit existierenden Ressourcen annotiert sind, kann das Framework diese auch auf aktualisierte Prozessmodelle abbilden. Für die Aktualisierung von Datenmodellen muss die Anwendung neu gestartet werden. Laufende Prozessinstanzen verwenden dann das aktualisierte Datenmodell.

A7) Die Verknüpfung von Prozess- und Ressourcenmodell soll über eine grafische Oberfläche stattfinden.
Sofern das verwendete BPMN Modellierungstool Datenobjekte unterstützt, kann die Verknüpfung ohne manuelle Anpassung der XML Datei durchgeführt werden.

A8) Das Framework soll die Verwendung verschiedener Workflow Engines und HTTP-Bibliotheken unterstützen.
Durch die modulare Architektur des Frameworks können unterschiedliche Engines und HTTP-Bibliotheken verwendet werden.

5 Evaluierung

Nachdem die Umsetzung des Frameworks beschrieben wurde, soll dieses im Folgenden Kapitel evaluiert werden. Das Ziel der Evaluation ist es, den Nutzen des Frameworks zu beurteilen. Dazu werden die Auswirkungen der generierten Schnittstelle auf den Clientcode, anhand eines Beispielszenarios, untersucht. Hierfür wird eine Beispielanwendung in zwei Ausführungen implementiert. Eine Version verwendet die generierte Schnittstelle, die andere Version verwendet die generische Schnittstelle einer Workflow Engine. Anschließend werden für den Vergleich der Schnittstellen Metriken für die verschiedenen Clients erhoben, welche die Auswirkung der Schnittstellen messbar machen sollen.

Dazu wird im Folgenden das Beispielszenario für den Vergleich vorgestellt. Anschließend werden die Auswirkungen der Schnittstellen auf die Clients mit Hilfe der ermittelten Metriken aufgezeigt.

5.1 Beispielszenario

Im Beispielszenario soll der Prozess für die Organisation von Wahlpflichtfächern an der Fakultät TI der HAW durch ein BPMS realisiert werden. Dieser Prozess umfasst das Erstellen der Fächer durch Dozenten, sowie die Auswahl und den Wechsel von Fächern durch Studierende. Hierfür soll ein Webclient für die Studierenden und Dozenten implementiert werden, mit dem die erforderlichen Aktivitäten ausgeführt werden können.

5.1.1 Geschäftsprozess

Das in Abbildung 5.1 abgebildete Prozessmodell beschreibt, wie Wahlpflichtfächer erstellt, gewählt und verteilt werden. Dieses basiert auf dem in [5] erarbeiteten Modell. Das ursprüngliche Model beschreibt den Prozess für ein Wahlpflichtfach. Damit dieser

für alle Studenten und Dozenten ausführbar ist, mussten einige Aktivitäten mit einem Multiinstanzmarker versehen werden.

Insgesamt gibt es drei Prozessbeteiligte: Dozenten, die HAW Hamburg sowie Studenten. Die Dozenten und Studenten sind als Blackbox dargestellt. Innerhalb der HAW sind das FSB, das Studierendeninformationssystem und die Homepage der HAW beteiligt. Letztere teilen sich im Prozessmodell eine Lane.

Der Prozess für die Wahlmodule wird durch das FSB gestartet, indem Dozenten durch eine Nachricht aufgefordert werden, WP-Angebote zu erstellen. Anschließend empfängt das FSB die Angebote der einzelnen Dozenten und prüft, ob es thematische Überschneidungen zwischen den Modulvorschlägen gibt. Bei Themen, die doppelt vorhanden sind, entscheidet das FSB, welches WP stattfinden soll.

Im nächsten Schritt finden drei Aktivitäten parallel statt. Die HAW stellt die Wahlpflichtangebote auf der Homepage ein, damit sich Studenten informieren können. Außerdem werden die Studierenden durch eine Nachricht darüber informiert, dass sie WPs wählen können. Dozenten, deren Module, aufgrund von Dopplung, nicht stattfinden, werden durch eine Nachricht darüber informiert.

Nachdem die drei Aktivitäten ausgeführt wurden, wartet der Prozess auf die WP-Auswahl der Studierenden. Wenn die Studierenden ihre Auswahl abgegeben haben, findet die Zuteilung von Studenten zu WPs statt. Hier kann es sein, dass Module aufgrund von zu wenig Teilnehmern ausfallen. Auch hier werden die Dozenten über den Ausfall informiert.

Zuletzt können Studierende ihr WP wechseln. Dazu müssen sie einen Wechselschein an das FSB senden. Der Wechsel eines WPs ist nicht das gesamte Semester möglich, weshalb an der Aktivität ein unterbrechendes Timerereignis angehängt ist.

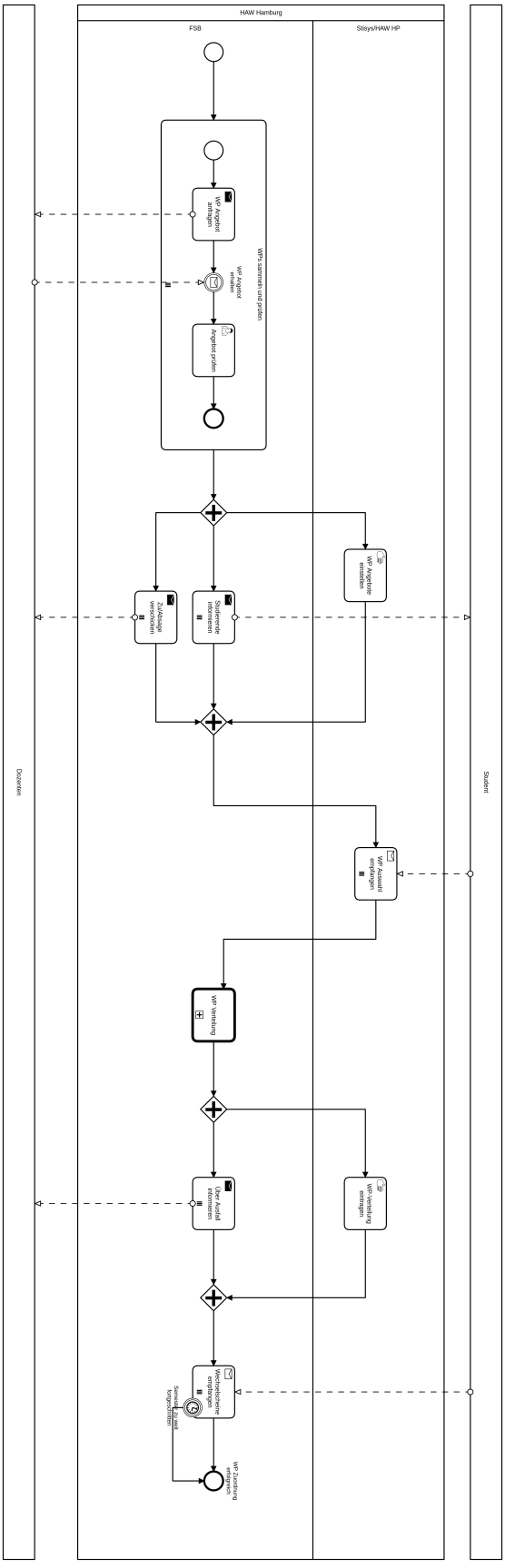


Abbildung 5.1: WP-Prozess

5.1.2 Datenmodell

Aus der Prozessbeschreibung lässt sich das in Abbildung 5.2 dargestellte Datenmodell ableiten.

Für jede Wahlphase wird eine Prozessinstanz des beschriebenen Prozesses ausgeführt. Die einzelnen Wahlphasen lassen sich durch das Semester, in dem sie stattfinden, unterscheiden. Das Ergebnis der Wahlphase ist eine Kursliste, dargestellt durch das Attribut "Kurse", welches durch die Komposition modelliert ist.

Während jeder Wahlphase werden Wahlpflichtangebote, dargestellt durch die Klasse "WPAngebot", von Dozenten erstellt. Dazu muss ein Name sowie eine Beschreibung für das Modul angegeben werden. Weiterhin muss bestimmt werden, wie viele Credits ein Student durch das Modul erhält. Das Attribut "duplikat" wird für die Bearbeitung der Module durch das FSB benötigt.

Studierende können während einer Wahlphase Module mit Prioritäten auswählen. Die Wahl eines einzelnen Studenten wird durch die Klasse WPAuswahl dargestellt, welche durch das Attribut "prioritäten" die priorisierten Wahlpflichtfächer sammelt.

Die von den Studierenden eingereichten Wechselscheine werden durch die Klasse Wechselschein dargestellt. Die Attribute beschreiben hier, aus welchem in welches Modul ein Student wechselt. Der beantragende Student wird wie in der WP-Auswahl durch die assoziierte Person dargestellt.

5.1.3 Umsetzung

Die Benutzerschnittstelle wurde mit dem Frontendframework Next.js[1] implementiert. Das Framework basiert auf der Javascript Bibliothek React[2] und ermöglicht serverseitiges Rendering, was die Trennung zwischen Anwendungslogik und Präsentation fördern soll.

Das zentrale Konzept des Frontendframeworks sind React-Komponenten. Eine Komponente stellt ein grafisches Element dar. Dies kann eine ganze Seite oder ein einzelner Button sein. Hier zeigt sich, dass eine Komponente aus anderen Komponenten, zusammengesetzt werden kann. Komponenten, die eine Seite darstellen, können durch vordefinierte Methoden ergänzt werden, welche die für das Rendering benötigten Daten liefern.

[1] https://nextjs.org/
[2] https://reactjs.org/

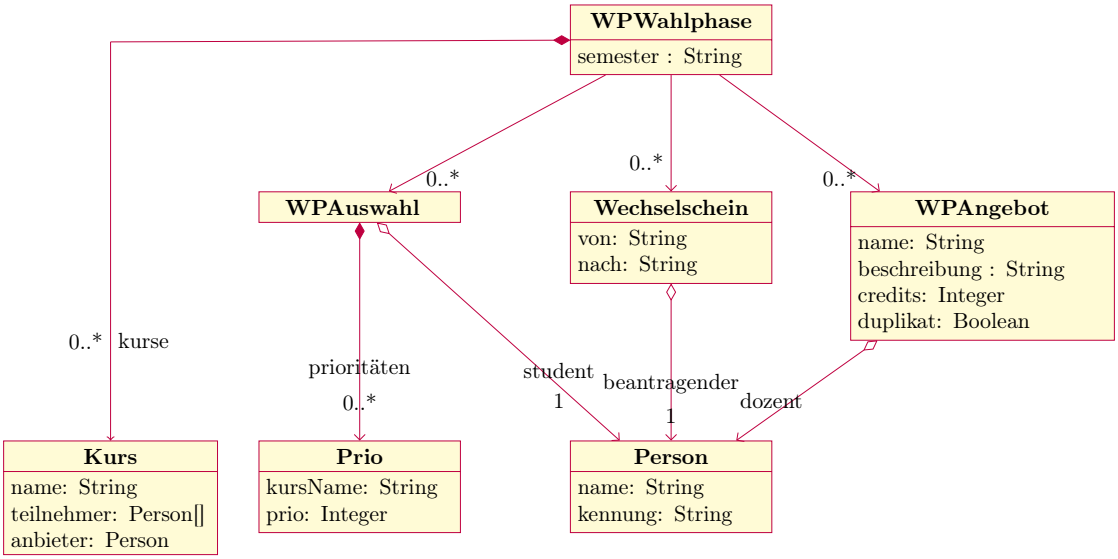


Abbildung 5.2: Datenmodell

Über die genauere Strukturierung einer Anwendung macht das Framework keine Vorgaben, solange diese mit dem Komponentenansatz vereinbar ist.

Die Anwendung für den WP Prozess besteht wie in Abbildung 5.3 gezeigt aus drei Seiten, also drei Grundkomponenten. Dabei wurde für jeden Prozessteilnehmer eine eigene Seite erstellt, auf der der Teilnehmer die für ihn im Prozess anfallenden Aufgaben erledigen kann. Dozenten können WPs erstellen, das FSB kann WPs auf Duplikate prüfen und absagen. Studierende können sich für WPs anmelden und Wechselscheine einreichen. Dadurch dass Studierende mehrere Aufgaben erledigen können, setzt sich die Studierendenkomponente aus zwei Teilkomponenten zusammen.

Für beide Implementierungen der Anwendungen wurde eine auf Services basierende Architektur gewählt. Jeder Service bietet die benötigten Methoden für ein Objekt aus dem Datenmodell. In Abbildung 5.3 zeigt sich, welche Komponenten welche Services nutzen.

Alle Komponenten müssen auf die aktuelle Wahlphase zugreifen, um auf die damit assoziierten Objekte zugreifen zu können. Daher verwendet jede Komponente den Wahlphasen Service. Außerdem muss jede Komponente den Angebote-Service nutzen. Die Komponente für Dozenten und das FSB erstellt und bearbeitet Angebote. Studierende müssen die angebotenen Module einsehen können.

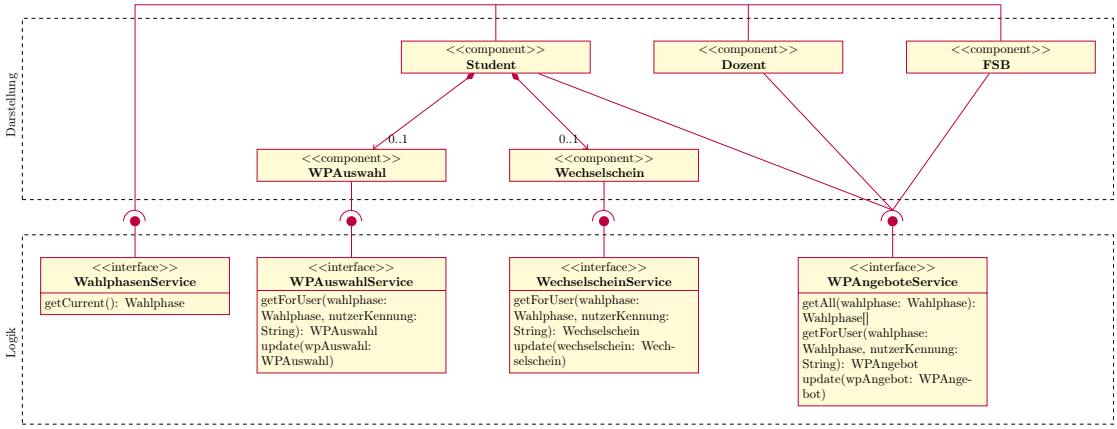


Abbildung 5.3: Komponenten der Frontendanwendung

Die WPAuswahl sowie die Wechselscheinkomponente benötigen lediglich die Services, die die entsprechenden Datentypen verwalten, um diese lesen und modifizieren zu können.

5.1.4 Implementierung mit generierter Schnittstelle

Für die Implementierung des Frontends mit generierter Schnittstelle wurde die Anwendung, wie in Abbildung 5.3 dargestellt, implementiert. Die Services lesen und modifizieren die Daten durch die Verwendung von REST-Aufrufen an die generierte Schnittstelle. Sie dienen als Adapter für die API, indem sie die Aufrufe kapseln und die Serverantworten in für die Komponenten darstellbare Datenobjekte umwandeln. Selbiges gilt auch für Fehlermeldungen.

5.1.5 Implementierung über die generische Schnittstelle der Workflow Engine

Um die Frontendanwendung mit der generischen REST API der Camunda Workflow Engine zu implementieren, wurde die Anwendung um BPMN-spezifische Services erweitert. Die Änderungen lassen sich der Abbildung 5.4 entnehmen.

Hierbei wurden Services zur Steuerung der Workflow Engine implementiert, um die REST-Aufrufe an die Engine zu kapseln. Die Aufteilung der Services sind den Komponenten der Camunda Engine (siehe Abbildung 2.14) entnommen. Dabei wurden nur

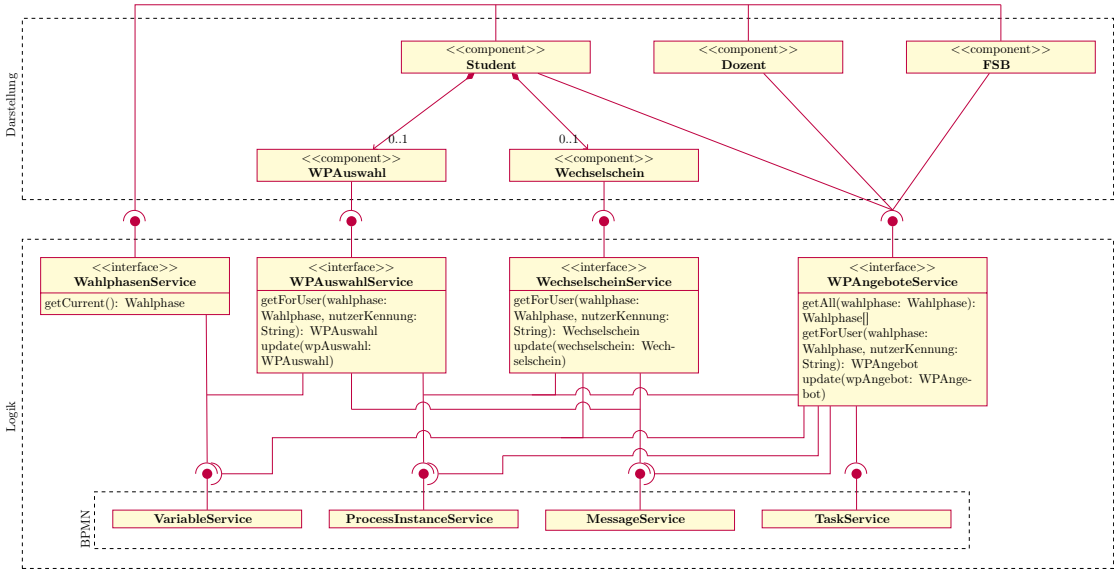


Abbildung 5.4: Komponenten der Frontendanwendung mit BPM Services

die Services und Methoden implementiert, die für die Anwendung benötigt werden. Die Aufgabe der Services ist in dieser Variante der Anwendung, Datenobjekte der Workflow Engine auf die fachlichen Daten abzubilden.

5.2 Auswirkungen auf Clientanwendungen

Die Schnittstellen können Auswirkungen auf Clients haben. Ein Vergleich der Komponenten der unterschiedlichen Implementierungen aus dem Beispielszenario macht dies bereits deutlich. In diesem Abschnitt sollen die Unterschiede der Anwendungen hinsichtlich der Softwarequalität genauer untersucht werden.

Die Qualität wird nach [16] anhand der Merkmale Wartbarkeit, Effizienz, Übertragbarkeit, Zuverlässigkeit, Funktionalität, und Benutzbarkeit bestimmt. Die Funktionalität, die Benutzbarkeit, sowie die Zuverlässigkeit sind bei beiden Varianten gleich, da diese Eigenschaften vor allem durch die Komponenten bestimmt werden, die sich bei beiden Anwendungen nicht unterscheiden. Daher werden die Clients hinsichtlich der Eigenschaften Wartbarkeit, Effizienz und Übertragbarkeit verglichen.

5.2.1 Wartbarkeit

Die Wartbarkeit von Software beschreibt, mit welchem Aufwand diese verändert werden kann, um zum Beispiel Fehler zu beheben, an neue Anforderungen angepasst zu werden, oder die Performanz zu erhöhen [4]. Neben der Dokumentation und der Spezifikation ist auch die Komplexität ein wichtiges Merkmal für die Wartbarkeit eines Systems, welches durch unterschiedliche Metriken auf diversen Abstraktionsebenen einer Anwendung gemessen werden kann [39].

Für die Bestimmung der Wartbarkeit der Beispielanwendung soll die Komplexität auf zwei Ebenen bestimmt werden: dem Entwurf der Anwendung sowie dem Quellcode.

Entwurfskomplexität

Die strukturelle Systemkomplexität beschreibt die Beziehungen zwischen den Modulen mit Hilfe des Fan-out-Maßes. Dieses Maß zählt die Anzahl ausgehender Verbindungen für ein Modul. Für die Berechnung der Komplexität wird das Quadrat der aufsummierten Fan-out-Beziehungen durch die Anzahl der Module einer Anwendung geteilt. Daraus ergibt sich das Verhältnis aus Modulverbindungen zu Modulen, wobei die Zahl der Verbindungen stärker gewichtet wird [41]. Ein hoher Wert zeigt an, dass es viele Abhän-

$$S(t) = \frac{\sum f(i)^2}{n}$$

$S(t)$: strukturelle Komplexität
$f(i)$: Fan-out für ein Modul
n : Anzahl der Module

Abbildung 5.5: Berechnung der Strukturellen Systemkomplexität

gigkeiten zwischen Modulen gibt, was sich auf die Verständlichkeit der Anwendung für Entwickler sowie die Testbarkeit auswirkt. Die Tabelle 5.1 zeigt die Berechnung für die Entwurfskomplexität beider Anwendungen. Hier zeigt sich, dass der Client, welcher die generische Schnittstelle verwendet, einen höheren Komplexitätswert als die Anwendung mit der fachlichen Schnittstelle hat. Durch die höhere Komplexität ist zu erwarten, dass Änderungen der Anwendung mit generischer Schnittstelle schwieriger durchzufüren sind, was für eine geringere Wartbarkeit spricht.

Modul	Fachliche Schnittstelle		Generische Schnittstelle	
	$f(i)$	$f(i)^2$	$f(i)$	$f(i)^2$
Student	2	4	2	4
Dozent	2	4	2	4
FSB	2	4	2	4
WPAuswahl	1	1	1	1
Wechselschein	1	1	1	1
WahlphasenService	0	0	1	1
WPAuswahlService	0	0	3	9
WechselscheinService	0	0	3	9
WPAngeboteService	0	0	5	25
VariableService	-	-	0	0
ProcessInstanceService	-	-	0	0
MessageService	-	-	0	0
TaskService	-	-	0	0
$\sum f(i)^2$		14		58
#Module		9		13
Komplexität		1.56		4.46

Tabelle 5.1: Entwurfskomplexität für die Beispielanwendungen

Code Komplexität

Mit Hilfe der McCabe Metrik soll die Code Komplexität gemessen werden. Hierbei wird die Anzahl der Verzweigungen des Kotrollflussgraphen der Anwendung ermittelt. Viele Pfade sorgen somit für höhere Komplexität, was sich negativ auf die Testbarkeit und Verständlichkeit auswirkt. Verschiedene Werkzeuge für die Überwachung der Code Qua-

$M = e - n + 2p$
e : Anzahl der Kanten im Kontrollgraphen
n : Anzahl der Knoten im Kontrollgraphen
p : Anzahl der Zusammenhangskomponenten im Graphen

Abbildung 5.6: Berechnung der McCabe Metrik

lität in Softwareprojekten unterstützen eine automatische Berechnung der Metrik, da

die manuelle Auswertung zu aufwändig ist. Für die Bestimmung des Wertes für die Beispielanwendung wurde das Tool Sonarqube[3] verwendet.

Abbildung 5.7: Ausschnitt aus dem Sonarqube Komplexitätsüberblick

Abbildung 5.7 zeigt einen Ausschnitt aus der Komplexitätsanalyse von Sonarqube. Dabei sind die Ordner und Dateien des Projekts auf der linken und die Komplexitätswerte auf der rechten Seite abgebildet. In den gezeigten Dateien sind die Interfaces für die Services definiert, weshalb diese keine zyklomatische Komplexität haben.

Im Ordner default sind die Implementierungen für die Services, welche die generische Schnittstelle der Camunda Workflow Engine verwenden. Zusammengezählt haben die Module in dem Ordner einen Wert von 47 für die McCabe Metrik.

Im Ordner generated sind die Implementierungen, welche die fachliche, generierte Schnittstelle verwenden. Diese kommen auf eine zyklomatische Komplexität von 20.

Je höher die McCabe Metrik für eine Anwendung ist, umso komplexer ist diese. Dies wirkt sich auf die Wartbarkeit aus. Ein Vergleich der McCabe Metrik für beide Anwendungen zeigt, dass die Clientimplementierung, welche die Schnittstelle der Workflow Engine verwendet, komplexer ist als die Implementierung der Anwendung, welche die generierte Schnittstelle verwendet.

5.2.2 Effizienz

Die Effizienz einer Anwendung beschreibt, wie diese mit Ressourcen umgeht. Dies lässt sich durch konkrete Größen wie den Speicherbedarf, die CPU Auslastung oder die Ladezeit in Sekunden messen.

[3] https://www.sonarqube.org/

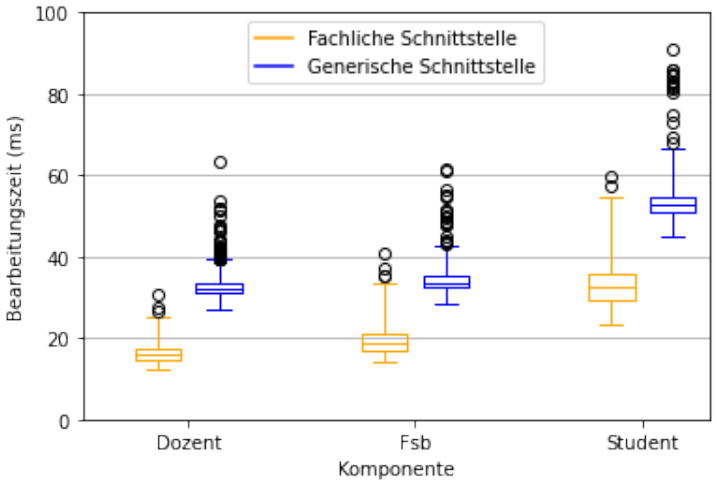


Abbildung 5.8: Vergleich der Bearbeitungszeiten

Bei der Beispielanwendung werden keine großen Datenmengen verwaltet und auch keine komplexen Berechnungen durchgeführt. Daher sind bei der Messung von Speicherbedarf oder CPU Nutzung keine großen Unterschiede zu erwarten. Worin sich die Anwendungen jedoch unterscheiden, ist die Menge der Netzwerkaufrufe, welche die unterschiedlichen Implementierungen der Services ausführen. Die Variante der Beispielanwendung, die direkt mit der Schnittstelle der Workflow Engine kommuniziert, muss mehrere Aufrufe an die REST-Schnittstelle der Engine machen, um auf die benötigten Ressourcen zugreifen zu können. Dabei hat jeder Aufruf an die REST API auch einen Aufruf an die Datenbank der Backendanwendung zur Folge. Sowohl Netzwerkverbindungen als auch Datenbankabfragen gelten als Flaschenhälse, die sich auf die Performanz einer Anwendung auswirken können.

Um den Effekt der Schnittstellen auf die Performanz der Clients zu untersuchen, wurde ein Lasttest mit Hilfe des Tools Gatling[4] durchgeführt. Dabei wurden über den Zeitraum von 10 Minuten 50 gleichzeitige Nutzer simuliert. Die Messung der Verarbeitungszeit der Services wird in den Methoden zur Datenbeschaffung der Komponenten durchgeführt, da hier die Aufrufe an die Services stattfinden. Zur Messung wurde die Process API[5] aus der NodeJs Standardbibliothek verwendet. Der Lasttest wurde in der ICC[6] durchgeführt. Dabei hatte jeder Pod einen CPU Kern und 2G RAM zur Verfügung.

[4] https://gatling.io/
[5] https://nodejs.org/api/process.html
[6] icc.informatik.haw-hamburg.de

Abbildung 5.8 zeigt die Verarbeitungszeiten der einzelnen Clientkomponenten für beide Implementierungsvarianten als Box-Plot. Dabei markieren die Grenzen der Box das obere und das untere Quartil. Die Whisker begrenzen die Werte innerhalb von $3 * IQR$, also die milden Ausreißer. In der Abbildung wird deutlich, dass die Bearbeitungszeit des Clients mit fachlicher Schnittstelle meist unter der Zeit des Clients mit generischer Schnittstelle liegt. Dies zeigt sich sowohl am Median als auch an den Quartilen. Darüber hinaus sind beim Client mit fachlicher Schnittstelle weniger extreme Ausreißer zu erkennen.

5.2.3 Übertragbarkeit

Die Übertragbarkeit von Software beantwortet die Frage, wie flexibel diese ist, um in neuen Umgebungen eingesetzt zu werden, und ob es Konflikte zu anderen Anwendungen geben kann. Durch den Einsatz von Containertechnologien sind moderne Anwendungen, was die Installation auf dem Host betrifft, leicht übertragbar.

Im Falle der Beispielanwendung kann die verwendete Workflow Engine als Teil der Umgebung betrachtet werden. Dadurch stellt die Austauschbarkeit der Workflow Engine auch einen Teil der Übertragbarkeit dar. Um dies zu messen, kann die Modulkopplung verwendet werden. Diese Metrik zeigt, wie stark die einzelnen Module einer Anwendung miteinander verknüpft sind, indem das Verhältnis von Modulverbindungen zu Modulen berechnet wird. Dies ermöglicht Rückschlüsse auf die Austauschbarkeit der Komponenten und somit auch der Workflow Engine.

$$\frac{\text{Anzahl der Modulverbindungen}}{\text{Anzahl der Module} + \text{Anzahl der Modulverbindungen}}$$

Abbildung 5.9: Berechnung der Modulkopplung

Erstrebenswert für eine Anwendung ist ein möglichst niedriger Wert. Nimmt die Modulkopplung einen Wert nahe eins an bedeutet dies, dass es im Vergleich zu der Anzahl der Module viele Verbindungen gibt. Dies deutet auf eine hohe Kopplung hin, was eine geringere Übertragbarkeit hinsichtlich der Engine bedeutet.

Tabelle 5.2 zeigt die Berechnung der Kopplung für beide Implementierungsvarianten. Hier zeigt sich, dass in der Anwendung mit generischer Schnittstelle ein höherer Kopplungsgrad vorliegt als in der Anwendung mit fachlicher Schnittstelle.

Metrik	Fachliche Schnittstelle	Generische Schnittstelle
# Modulverbindungen	9	13
# Module	10	22
Kopplung	0.47	0.59

Tabelle 5.2: Modulkopplung für die Beispielanwendungen

5.3 Diskussion

Der Vergleich der Metriken zeigt, dass sich die Softwarequalität durch die Verwendung der generierten, fachlichen Schnittstelle verbessert. Dies zeigt sich anhand der Wartbarkeit, die mit Hilfe der Komplexität auf Code- und auf Modulebene untersucht wurde. Auch die Effizienz und die Übertragbarkeit der Anwendung unterstützen die These, dass die generierte, fachliche Schnittstelle zur Softwarequalität beiträgt. Hierbei ist zu betonen, dass es sich bei der Anwendung um ein einzelnes Beispiel handelt. Für aussagekräftigere Ergebnisse ist eine Evaluation des entwickelten Frameworks in größeren Fallstudien notwendig.

Außerdem ist für die Metriken, welche anhand des Entwurfs der Beispielanwendung ermittelt werden, bestreitbar, ob der Entwurf für die Anwendung mit generischer Schnittstelle angemessen ist. So lässt sich die Kopplung beispielsweise reduzieren, indem sämtliche Funktionalitäten durch ein einziges Workflow Engine Interface implementiert werden. Auch wenn sich durch solche Änderungen einige Metriken optimieren lassen, ist es im Sinne von sauberen Architekturen, die Komponenten nach Verantworlichkeiten aufzuteilen. Darüber hinaus haben solche Veränderungen auch Einfluss auf den Code, was die Wartbarkeit auf dieser Ebene verändern kann.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde die Entwicklung eines Frameworks zur Generierung fachlicher Schnittstellen für Workflow Engines beschrieben. Die Lösung beschränkt sich auf Engines, welche BPMN Modelle ausführen können. Dazu wurde ein Metamodell entwickelt, welches Erweiterungen für die BPMN vorsieht und aufzeigt, wie Prozessmodelle mit Datenmodellen verknüpft werden können. Das Framework wurde mit der Anforderung entwickelt, dass es auf unterschiedliche Workflow Engines und HTTP-Bibliotheken anwendbar ist. Dies wird durch eine modulare Architektur erreicht, welche die austauschbaren Komponenten durch eine Abstraktionsschicht vom Anwendungskern trennt.

6.1 Fazit

Die Evaluation zeigt, dass die vom Framework generierten fachlichen Schnittstellen zur Qualität der Clients beitragen können. Dies wurde anhand eines Beispiels für die Qualitätseigenschaften Wartbarkeit, Effizienz und Übertragbarkeit gezeigt.

Die Wartbarkeit der Anwendung wird gefördert, indem die Komplexität der Software, welche die fachliche Schnittstelle verwendet, reduziert wird. Dies zeigt sich sowohl für den Softwareentwurf als auch für den Anwendungscode. Bei einem Vergleich der Bearbeitungszeit von Anfragen, welche Clients mit den unterschiedlichen Schnittstellen benötigen, wird sichtbar, dass die Anwendung, welche die vom Framework generierte Schnittstelle verwendet, weniger Zeit für die Kommunikation mit dem Backend benötigt. Zuletzt wurde gezeigt, dass die fachliche Schnittstelle die Kopplung zur Workflow Engine reduziert, was die Übertragbarkeit der Clientanwendungen erhöht.

6.2 Ausblick

In der Evaluation wurde gezeigt, dass sich Anwendungen für automatisierte Geschäftsprozesse mit den vom Framework angebotenen Funktionalitäten realisieren lassen. Für weiterführende Arbeiten ist die Implementierung zusätzlicher Funktionen, welche die Entwicklungsarbeit erleichtern sollen, denkbar. Die BPMN bietet beispielsweise Mechanismen für User Tasks, um diese automatisch bestimmten Gruppen, wie Abteilungen oder einzelnen Personen, zuzuordnen. Aktuell muss im Framework eine Rechteüberprüfung für Anfragen manuell im Validierungsinterface implementiert werden. Workflow Engines bieten meist standardmäßig Authentifizierungs- und Autorisierungsfunktionalitäten für die Nutzer. Mit den Informationen über Nutzer, die bereits in BPMN Modellen festgehalten werden können, ergänzt durch die Authentifizierungsdienste der Engine, ist es naheliegend, dass das Framework Autorisierung und Authentifizierung unterstützen soll. Darüber hinaus gibt es Ansätze wie der in [20], die eine Modellierung von Rechten auch visuell mit Hilfe der UML unterstützen. Die Implementierung des Features würde eine Analyse von Authentifizierungsmechanismen gängiger Workflow Engines voraussetzen. Anhand dieser Analyse kann das Interface zur Abstraktion von Workflow Engines mit Methoden zur Authentifizierung und Autorisierung so erweitert werden, dass weiterhin möglichst viele Workflow Engines mit dem Framework verwendet werden können.

Ein ebenfalls interessantes Feature ist die automatische Generierung des Ressourcenbaumes, welcher aktuell noch manuell modelliert werden muss. Aus den Arbeiten [36, 33] zeigt sich, dass aus den BPMN Elementen eine Hierarchie gebildet werden kann. Diese Hierarchie kann auf die verknüpften Ressourcen übertragen werden, um die URIs zu generieren. Aus [21] geht ebenfalls hervor, dass für Objekte in objektzentrierten Prozessen solch eine Hierarchie gebildet werden kann. Für die Generierung der URIs wird nicht nur die Hierarchie der Objekte benötigt, sondern auch die Multiplizität der Assoziationen. Diese können ermittelt werden, indem die Prozesse auf Schleifen und bedingte Ausführungen untersucht werden.

Neben neuen Funktionalitäten, bietet auch die Evaluation des Frameworks Potential für weiterführende Arbeiten. In dieser Arbeit wurde gezeigt, wie sich die Verwendung des Frameworks auf die Softwarequalität der Clients auswirkt. Allerdings wurde noch nicht untersucht, wie sich das Framework in den Softwareentwicklungsprozess einfügen lässt. Darüber hinaus muss die Bedienbarkeit des Frameworks sowie die Verständlichkeit des darunterliegenden Konzeptes analysiert werden. Dies lässt sich durch Fallstudien, in

denen das Framework eingesetzt werden soll, realisieren. Für die Ermittlung der Bedienbarkeit und Verständlichkeit können Methoden wie Fragebögen, zum Beispiel die System Usability Scale [7], oder Interviews verwendet werden.

Literaturverzeichnis

[1] *Activiti User Guide*. v6.0.0. – URL https://www.activiti.org/userguide/#apiVariables. – Zugriffsdatum: 2020-07-16

[2] *jBPM User Guide*. v4. – URL https://docs.jboss.org/jbpm/v4/userguide/html_single/#variables. – Zugriffsdatum: 2020-07-16

[3] *Process Variables | docs.camunda.org*. 7.5. – URL https://docs.camunda.org/manual/7.5/user-guide/process-engine/variables/. – Zugriffsdatum: 2020-07-16

[4] IEEE Standard for Software Configuration Management Plans. In: *IEEE Std 828-1983* (1983), S. 1–10

[5] Brinkmeier, Andreas: *Werkzeugunterstützte Analyse und Optimierung von Prozessen einer Universität*. 2017. – URL https://reposit.haw-hamburg.de/handle/20.500.12738/8125

[6] Brocke, Jan vom (Hrsg.) ; Rosemann, Michael (Hrsg.): *Handbook on Business Process Management 1: Introduction, Methods, and Information Systems*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2015. – ISBN 978-3-642-45099-0 978-3-642-45100-3

[7] Brooke, J: *SUS: A quick and dirty usability scale. Teoksessa PW Jordan, B. Thomas, BA Weerdmeester & IL McClelland (toim.) Usability Evaluation in Industry*. 1996

[8] Davenport, Thomas H.: *Process innovation: reengineering work through information technology*. Boston, Mass : Harvard Business School Press, 1993. – ISBN 978-0-87584-366-7

[9] Dumas, Marlon ; La Rosa, Marcello ; Mendling, Jan ; Reijers, Hajo A.: *Fundamentals of Business Process Management*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2018. – ISBN 978-3-662-56508-7 978-3-662-56509-4

[10] Ed-douibi, Hamza ; Izquierdo, Javier Luis C. ; Gómez, Abel ; Tisi, Massimo ; Cabot, Jordi: EMF-REST: generation of RESTful APIs from models. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. Pisa, Italy : Association for Computing Machinery, April 2016 (SAC '16), S. 1446–1453. – ISBN 978-1-4503-3739-7

[11] Fielding, Roy T. ; Taylor, Richard N.: *Architectural Styles and the Design of Network-Based Software Architectures*, Dissertation, 2000

[12] Geiger, Matthias ; Harrer, Simon ; Lenhard, Jorg ; Wirtz, Guido: On the Evolution of BPMN 2.0 Support and Implementation. In: *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. Oxford : IEEE, März 2016, S. 101–110

[13] Giessler, Pascal ; Gebhart, Michael ; Sarancin, Dmitrij ; Steinegger, Roland ; Abeck, Sebastian: Best Practices for the Design of RESTful Web Services. In: *Proceedings - International Conference on Software Engineering* 10 (2015), November, S. 392–397

[14] Hanák, Bc D.: *GraphQL as modern access to jBPM process engine*, Masaryk University, Diplomarbeit, 6 2019. – 80 S

[15] Haupt, Florian ; Karastoyanova, Dimka ; Leymann, Frank ; Schroth, Benjamin: A Model-Driven Approach for REST Compliant Services. In: *2014 IEEE International Conference on Web Services*. Anchorage, AK, USA : IEEE, Juni 2014, S. 129–136. – ISBN 978-1-4799-5054-6 978-1-4799-5053-9

[16] Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models / International Organization for Standardization. Geneva, CH, März 2011. – Standard

[17] JennyCooper: *REST APIs can reflect business processes*. September 2019. – URL https://developer.ibm.com/watsonhealth/2019/09/23/rest-apis-can-reflect-business-processes/. – Zugriffsdatum: 2020-07-08. – Library Catalog: developer.ibm.com Section: Cúram Social Program Management

[18] Kelly, Mike: JSON Hypertext Application Language / IETF Secretariat. May 2016 (draft-kelly-json-hal-08). – Internet-Draft

[19] Kluza, Krzysztof ; Kaczor, Krzysztof ; Nalepa, Grzegorz J. ; Ślażyński, Mateusz: Opportunities for Business Process semantization in open-source process

execution environments. In: *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, September 2015, S. 1307–1314

[20] Koch, Manuel ; Parisi-Presicce, Francesco: UML specification of access control policies and their formal verification. In: *Software & Systems Modeling* 5 (2006), Oktober, Nr. 4, S. 429–447. – URL https://doi.org/10.1007/s10270-006-0030-z

[21] Künzle, Vera ; Reichert, Manfred: Towards Object-Aware Process Management Systems: Issues, Challenges, Benefits. In: Halpin, Terry (Hrsg.) ; Krogstie, John (Hrsg.) ; Nurcan, Selmin (Hrsg.) ; Proper, Erik (Hrsg.) ; Schmidt, Rainer (Hrsg.) ; Soffer, Pnina (Hrsg.) ; Ukor, Roland (Hrsg.): *Enterprise, Business-Process and Information Systems Modeling* Bd. 29. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, S. 197–210

[22] Künzle, Vera ; Reichert, Manfred: PHILharmonicFlows: towards a framework for object-aware process management. In: *Journal of Software Maintenance and Evolution: Research and Practice* 23 (2011), Juni, Nr. 4, S. 205–244. – ISSN 1532060X

[23] Künzle, Vera ; Reichert, Manfred: Striving for Object-Aware Process Support: How Existing Approaches Fit Together. In: Aberer, Karl (Hrsg.) ; Damiani, Ernesto (Hrsg.) ; Dillon, Tharam (Hrsg.): *Data-Driven Process Discovery and Analysis* Bd. 116. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, S. 169–188. – Series Title: Lecture Notes in Business Information Processing

[24] Laitkorpi, Markku ; Koskinen, Johannes ; Systa, Tarja: A UML-based Approach for Abstracting Application Interfaces to REST-like Services. In: *2006 13th Working Conference on Reverse Engineering*, Oktober 2006, S. 134–146. – ISSN: 2375-5369

[25] Lenhard, Jörg ; Ferme, Vincenzo ; Harrer, Simon ; Geiger, Matthias ; Pautasso, Cesare: Lessons Learned from Evaluating Workflow Management Systems. In: Braubach, Lars (Hrsg.) ; Murillo, Juan M. (Hrsg.) ; Kaviani, Nima (Hrsg.) ; Lama, Manuel (Hrsg.) ; Burgueño, Loli (Hrsg.) ; Moha, Naouel (Hrsg.) ; Oriol, Marc (Hrsg.): *Service-Oriented Computing – ICSOC 2017 Workshops*. Cham : Springer International Publishing, 2018, S. 215–227

[26] Liskin, Olga ; Singer, Leif ; Schneider, Kurt: Teaching Old Services New Tricks: Adding HATEOAS Support as an Afterthought. In: *Proceedings of the Second International Workshop on RESTful Design*. New York, NY, USA : Association for Computing Machinery, 2011 (WS-REST '11), S. 3–10

[27] Meyer, Andreas ; Smirnov, Sergey ; Weske, Mathias: *Data in business processes*. Potsdam : Univ.-Verl. Potsdam, 2011 (Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam 50). – ISBN 978-3-86956-144-8

[28] Neumann, Andy ; Laranjeiro, Nuno ; Bernardino, Jorge: An Analysis of Public REST Web Service APIs. In: *IEEE Transactions on Services Computing* (2018), S. 1–1. – Conference Name: IEEE Transactions on Services Computing. – ISSN 1939-1374

[29] Newman, Sam ; Lorenzen, Knut (Hrsg.): *Microservices Konzeption und Design*. 1. Aufl. mitp-Verl., 2015

[30] Nielsen, Henrik ; Mogul, Jeffrey ; Masinter, Larry M. ; Fielding, Roy T. ; Gettys, Jim ; Leach, Paul J. ; Berners-Lee, Tim: *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. Juni 1999. – URL https://rfc-editor.org/rfc/rfc2616.txt

[31] Nikaj, Adriatik: *REST-ChoreografienRestful choreographies*, Universität Potsdam, Dissertation, 2019. – URL https://publishup.uni-potsdam.de/43890. – Zugriffsdatum: 2020-07-07

[32] Nottingham, Mark ; Sayre, Robert: The Atom Syndication Format / RFC Editor. RFC Editor, December 2005 (4287). – RFC. – URL http://www.rfc-editor.org/rfc/rfc4287.txt. http://www.rfc-editor.org/rfc/rfc4287.txt. – ISSN 2070-1721

[33] Oberhauser, R.: A Hypermedia-Driven Approach for Adapting Processes via Adaptation Processes. In: *2015 8th International Conference on Advanced Software Engineering Its Applications (ASEA)*, November 2015, S. 73–80

[34] OMG: Business Process Model and Notation (BPMN), Version 2.0.2 / Object Management Group. OMG, Dezember 2013. – Forschungsbericht. – URL http://www.omg.org/spec/BPMN/2.0.2.

[35] OMG: Object Constraint Language, Version 2.0.2 / Object Management Group. OMG, Dezember 2014. – Forschungsbericht. – URL https://www.omg.org/spec/OCL/About-OCL/.

[36] Pautasso, Cesare: BPMN for REST. In: Dijkman, Remco (Hrsg.) ; Hofstetter, Jörg (Hrsg.) ; Koehler, Jana (Hrsg.): *Business Process Model and Notation*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, S. 74–87. – ISBN 978-3-642-25160-3

[37] Pietruck, Dennis: *Konzeption von fachlichen REST APIs für Prozessanwendungen*. 2018. – URL https://edoc.sub.uni-hamburg.de/haw/volltexte/2018/4430/pdf/ba_pietruck.pdf

[38] Reijers, H.A. ; Vanderfeesten, I. ; Aalst, W.M.P. van der: The effectiveness of workflow management systems: A longitudinal study. In: *International Journal of Information Management* 36 (2016), Februar, Nr. 1, S. 126–141

[39] Riaz, M. ; Mendes, E. ; Tempero, E.: A systematic review of software maintainability prediction and metrics. In: *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009, S. 367–377

[40] Ryndina, Ksenia ; Küster, Jochen M. ; Gall, Harald: Consistency of Business Process Models and Object Life Cycles. In: Kühne, Thomas (Hrsg.): *Models in Software Engineering* Bd. 4364. Berlin, Heidelberg : Springer Berlin Heidelberg, 2007, S. 80–90

[41] Sneed, Harry M. 1. ; Seidl, Richard 1. (Hrsg.): *Software in Zahlen die Vermessung von Applikationen*. Hanser, 2010. – URL http://d-nb.info/1002464056/04

[42] Starke, Gernot 1. ; Verlag, Carl H. (Hrsg.): *Effektive Softwarearchitekturen ein praktischer Leitfaden*. 9., überarbeitete Auflage. Hanser, 2020 (Hanser eLibrary)

[43] Wang, Bo ; Rosenberg, Doug ; Boehm, Barry W.: Rapid realization of executable domain models via automatic code generation. In: *2017 IEEE 28th Annual Software Technology Conference (STC)*, September 2017, S. 1–6

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Masterarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Entwicklung und Evaluation eines Frameworks zur Generierung fachlicher Schnittstellen für Geschäftsprozesse

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original