
ulm university universität
**u**ulm

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

# Design und Implementierung eines Webservices für die automatisierte Generierung von BPMN 2.0 Prozessmodellen

Bachelorarbeit an der Universität Ulm

**Vorgelegt von:**
Leon Ahmadi-Moghaddam
leon.ahmadi-moghaddam@uni-ulm.de
1034441

**Gutachter:**
Prof. Dr. Manfred Reichert

**Betreuer:**
Michael Winter

2020

Fassung 29. Oktober 2020



Satz: PDF-LaTeX 2ε

# Kurzfassung

BPMN 2.0 ist ein branchenübergreifender Standard für die Modellierung von Prozessen. Neben der grafischen Repräsentation der Modelle definiert der Standard außerdem eine maschinell verarbeitbare XML-Repräsentation. Im Rahmen von Studienarbeiten zu der Verständlichkeit von BPMN-Modellen werden syntaktisch unterschiedliche Diagramme benötigt. Damit diese nicht wiederkehrend händisch modelliert werden müssen, wurde der BPMN-Generator zur automatisierten Generierung von Prozessmodellen auf der Grundlage gewisser Benutzereingaben (Anzahl der Aktivitäten, Gateways, Events, etc.) entwickelt. Bei dem BPMN-Generator handelt es sich um einen Express.js-Webservice, der in einer Node.js-Umgebung ausgeführt werden kann. Die Datenhaltung erfolgt über eine MongoDB-Instanz und als Client dient ein gewöhnlicher Webbrowser. Die entwickelte Software zeichnet sich durch einen modularisierten Entwurf aus, wodurch das Produkt einen hohen Grad an Wartbarkeit und Erweiterbarkeit erhält. Im Vergleich zu bisherigen Modellierungsumgebungen wie dem Signavio Process Manager oder bpmn.io kann sich der BPMN-Generator durch die Funktionalität der automatisierten Generierung hervortun und bietet somit einen anwendbaren Mehrwert.

# Inhaltsverzeichnis

# Abkürzungsverzeichnis

**BPM** Business Process Management (Geschäftsprozess-Management)

**BPMN** Business Process Model and Notation

**OMG** Object Management Group

**DBMS** Datenbankmanagementsystem

# 1 Einleitung

Das Business Process Management (Geschäftsprozess-Management) (BPM) beschreibt die Disziplin der systematischen Erfassung, Analyse und Optimierung von Geschäftsprozessen [7]. Ein Geschäftsprozess definiert eine Menge von Tätigkeiten, die in einer bestimmten zeitlichen Abfolge und unter Beachtung von Ereignissen und Bedingungen ausgeführt werden, um ein gewisses Ziel zu erreichen [7]. Im Hinblick auf das ökonomische Prinzip liegt die Relevanz des BPM und somit das unternehmerische Interesse, Geschäftsprozesse zu optimieren, nahe.

Die Prozessoptimierung folgt aus theoretischer Sicht dem Modell des BPM-Lebenszykluses. Dieser definiert mehrere Phasen, die zyklisch wiederkehrend ausgeführt werden und so einen bestehenden Geschäftsprozess iterativ verbessern. Abbildung 1.1 zeigt die Grundstruktur des Lebenszykluses nach [6]. Gemäß dieses Modells müssen Prozesse in der Organisation zunächst identifiziert und voneinander abgetrennt werden. Im nächsten Schritt, der Prozess-Entdeckung, werden die Ist-Zustände der Prozesse erfasst und als Modell festgehalten. Anschließend werden Schwachstellen der bisherigen Prozesse auf der Grundlage der zuvor modellierten Prozesse identifiziert und dokumentiert. In der Phase der konkreten Prozessoptimierung werden mögliche Änderungen erarbeitet, die den Prozess verbessern. Diese werden in einem Soll-Prozessmodell erfasst. Schließlich werden diese Soll-Prozesse in der Organisation eingeführt und der Erfolg der Optimierungen wird überwacht, woraufhin ggf. eine weitere Iteration des Lebenszykluses startet. [6]

Die Prozessoptimierung bedarf also einer formalen Erfassung des Soll- und des Ist-Zustandes mit Hilfe von Prozessmodellierungssprachen. Sie dienen als Kommunikations- und Bewertungsgrundlage und spezifizieren das gewünschte Ergebnis. Darüber hinaus ist es möglich, Prozessmodelle von Softwaresystemen einlesen zu lassen, um den Prozess vollständig oder zu Teilen zu automatisieren. Die Object Management Group (OMG) definiert mit der Business Process Model and Nota-

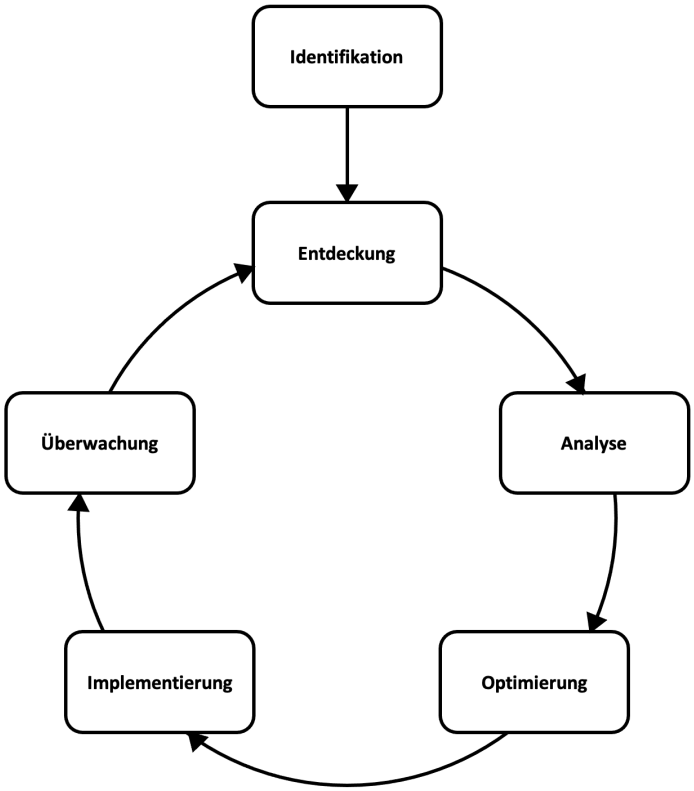


Abbildung 1.1: Der BPM-Lebenszyklus nach [6].

tion (BPMN) 2.0 einen Modellierungs-Standard [4]. Konkret handelt es sich bei BPMN 2.0 um eine grafische Modellierungssprache, die durch ein XML-basiertes Format technisch implementiert wird. BPMN bietet durch die grafische Modellierung den Vorteil, dass es leicht verständlich ist und somit interdisziplinär als Werkzeug anerkannt wird, sowie die Möglichkeit der maschinellen Verarbeitung durch die Definition des Standards.

Zusammenfassend zeigt sich, dass das BPM mit den zugehörigen Modellierungssprachen ein großes und praxisrelevantes wissenschaftliches Feld aufspannt. Weiterführende grundlegende Informationen und theoretische Hintergründe lassen sich [7] und [4] entnehmen.

## 1.1 Problembeschreibung

Die Verständlichkeit von Prozessmodellen ist im wissenschaftlichen Bereich des BPM ein zentrales Thema. Grafische Modellierungsansätze ermöglichen grundsätzlich ein leichteres Verständnis, doch auch etablierte Modellierungssprachen wie BPMN können fehlinterpretiert werden. Es ist demnach erforderlich, durch wissenschaftliche Arbeiten Schwachstellen der Modellierung zu identifizieren. So haben beispielsweise Mendling et al. auf der Grundlage von empirischen Studien die sieben folgenden Richtlinien definiert, anhand derer bei dem Entwurf von Prozessmodellen verständlichere Ergebnisse erzielt werden sollen [8]:

1. Reduziere die Anzahl der Elemente im Modell
2. Reduziere die Anzahl der Flüsse / Pfade eines Elements
3. Verwende nur ein Start- und ein End-Event
4. Verbessere die Struktur des Modells
5. Vermeide OR-Elemente
6. Benenne Aktivitäten nach dem Verb-Objekt Muster
7. Zerlege das Modell, wenn es mehr als 50 Elemente hat

Außerdem zeigen die Ergebnisse des Eye-tracking Experiments von Zimoch et al., dass unerfahrene Anwender Schwierigkeiten bei dem Verstehen von BPMN 2.0 Modellen haben, wohingegen Experten diese deutlich effizienter begreifen [12]. Für die Durchführung solcher Studien ist es bislang notwendig, die Prozessmodelle manuell zu erstellen. Diese Modellgenerierung soll durch eine Softwareanwendung automatisiert werden.

## 1.2 Implementierungsgegenstand

Zum Zwecke der automatisierten Generierung von BPMN 2.0-Prozessmodellen wurde im Rahmen dieser Arbeit eine Webanwendung entwickelt. Der Benutzer konfiguriert Diagramm-Parameter, auf deren Grundlage die Software ein oder mehrere BPMN 2.0-Diagramme generiert. Diese werden in einer Datenbank gespeichert

und können von dem Anwender in die Dateiaustauschformate PNG, PDF, XML und SVG exportiert werden. Die im Zuge des Softwareentwicklungsprozesses entstandenen Ergebnisse der Anforderungsanalyse, des Entwurfs und der Implementierung werden in der folgenden Ausarbeitung vorgestellt. Die entwickelte Anwendung wird namentlich als 'BPMN-Generator' ausgewiesen.

## 1.3 Struktur der Arbeit

In der fortlaufenden Ausarbeitung werden Entscheidungen und Überlegungen des Softwareentwicklungsprozesses erläutert und begründet. Zunächst werden mit den funktionalen und nichtfunktionalen Anforderungen an das System die Ergebnisse der Anforderungsanalyse vorgestellt. Anschließend wird in Kapitel 3 die Wahl der eingesetzten Technologien begründet. Kapitel 4 diskutiert die Entwurfsentscheidungen, insbesondere im Hinblick auf die Systemstrukturierung, den Architekturentwurf sowie den Feinentwurf der Software. Hierfür werden veranschaulichende Diagramme herangezogen und Bezug auf die einzelnen Komponenten genommen. Das darauf folgende Kapitel 5 stellt die Implementierung der wichtigsten Bestandteile vor und gibt somit auf Codeebene einen Überblick über das Projekt. Daraufhin werden die in Kapitel 2 vorgestellten Anforderungen an das Projekt auf der Grundlage der abgeschlossenen Softwareentwicklung reflektiert und eventuelle Abweichungen begründet. Mit Kapitel 7 folgt eine Vorstellung verwandter Arbeiten, insbesondere ähnlicher Modellierungsumgebungen für BPMN 2.0 Diagramme, die anschließend vom BPMN-Generator abgegrenzt werden. Schlussendlich werden die erarbeiteten Ergebnisse zusammengefasst und es wird ein Ausblick auf die Weiterentwicklung des Services gegeben.

# 2 Anforderungsanalyse

Ein essenzieller Bestandteil des Softwareentwicklungsprozesses ist die Anforderungsanalyse. In dieser Phase werden sowohl funktionale als auch nichtfunktionale Anforderungen an das zu entwickelnde Softwaresystem erarbeitet und festgehalten. Diese dienen als Kommunikationsgrundlage für die Erwartungen des Auftraggebers und setzen somit den Rahmen für den Software-Entwurf und die anschließende Implementierung. Funktionale Anforderungen betreffen die Funktionalität des zu entwickelnden Systems, während nichtfunktionale Anforderungen auf Qualitäts- und Entwicklungsaspekte eingehen. Die Anforderungen sollen präzise und vollständig formuliert werden, um eine Bewertungsgrundlage für die Software zu schaffen.

Trotz der gewissenhaften Definition und Formulierung von funktionalen Anforderungen in der Analysephase, können sich diese im Laufe des dynamischen Softwareentwicklungsprozesses verändern. In agilen Vorgehensmodellen ist es deshalb üblich, die Anforderungen iterativ zu definieren. Die Erreichung der in den folgenden Abschnitten definierten Anforderungen für den BPMN-Generator wird deshalb in Kapitel 6 analysiert und Abweichungen werden ggf. begründet.

## 2.1 Funktionale Anforderungen

Bei der zu entwickelnden Software handelt es sich um einen Webservice, dem der Benutzer über eine Webbrowser-Schnittstelle Parameter für die Generierung von BPMN-Diagrammen übergeben kann. Die generierten Diagramme sollen anschließend persistiert werden und der Nutzer soll diese in unterschiedliche Austauschformate exportieren können. Die erarbeiteten Anforderungen sind im Folgenden gelistet.

| Name | Beschreibung |
|---|---|
| FA-01: Plattform | Die Anwendung stellt ein Web-Interface bereit, welches über einen Browser erreicht werden kann. |
| FA-02: Benutzerschnittstelle | Die Benutzerschnittstelle umfasst zwei Ansichten:<br>• Meine Modelle: Hier kann der Benutzer Modelle, die in einer Datenbank gespeichert sind, einsehen, herunterladen und löschen.<br>• Generator: Hier kann der Benutzer Parameter setzen (siehe FA-03), auf deren Grundlagen BPMN-Modelle generiert werden. |
| FA-03: Eingabeparameter des Generators | Zum Generieren der BPMN-Modelle setzt der Benutzer Parameter, auf deren Grundlage die BPMN-Modelle generiert werden. Mindestens von der Software unterstützt werden:<br>• Anzahl der Aktivitäten<br>• Anzahl der XOR- und AND-Gateways<br>• Anzahl der Start- und End-Events<br>• Anzahl der zu generierenden Modelle |
| FA-04: Valide Syntax | Alle generierten BPMN-Modelle sind gemäß des BPMN 2.0 Standards in syntaktisch korrekter Form. Sofern es nicht möglich ist, die vom Benutzer definierte Anzahl unterschiedlicher Diagramme zu generieren (z.B. aufgrund der Restriktionen durch die anderen Parameter), wird dem Benutzer eine Fehlermeldung ausgegeben. |

Tabelle 2.1: Funktionale Anforderungen an den BPMN-Generator.

| Name | Beschreibung |
|---|---|
| FA-05: Persistenz | Die von dem Generator generierten Modelle können wahlweise in einer Datenbank gespeichert werden. Alle gespeicherten Modelle sind über die Ansicht “Meine Modelle” einsehbar (siehe FA-02). Alternativ können die generierten Modelle auf dem lokalen Gerät gespeichert werden. |
| FA-06: Ausgabeformat | Die generierten Modelle können in folgende Formate exportiert werden:<br>• XML<br>• PDF<br>• PNG |

Tabelle 2.1: Funktionale Anforderungen an den BPMN-Generator.

## 2.2 Nichtfunktionale Anforderungen

Da sich ein gelungenes Softwareprodukt ebenfalls durch weitere, nicht die Funktionalität betreffende Aspekte auszeichnet, werden darüber hinaus nichtfunktionale Anforderungen definiert. Hierbei sind die Robustheit, Wartbarkeit, Erweiterbarkeit sowie die Dokumentation und Tests meist zentraler Gegenstand der Betrachtung.

| Name | Beschreibung |
|---|---|
| NA-01: Wartbarkeit und Erweiterbarkeit | Die Software ist zu einem hohen Grad modularisiert und ist mit Rücksicht auf Wartbarkeit und Erweiterbarkeit entworfen. |
| NA-02: Robustheit | Fehlerhafte Eingaben werden vom System abgefangen und dem Benutzer wird eine Fehlermeldung angezeigt. |

Tabelle 2.2: Nichtfunktionale Anforderungen an den BPMN-Generator.

| Name | Beschreibung |
|---|---|
| NA-03: Behandlung von Laufzeitfehlern | Bei einem Auftreten von Laufzeitfehlern behandelt das System diese angemessen. Sie werden im Log dokumentiert und dem Benutzer wird eine Fehlermeldung angezeigt. |
| NA-04: Testabdeckung | Unit-Tests unterstützen das Entwickeln eines robusten Systems. |
| NA-05: Dokumentation | Der Quellcode sowie die Schnittstellen des Systems werden dokumentiert. |

Tabelle 2.2: Nichtfunktionale Anforderungen an den BPMN-Generator.

Diese nichtfunktionalen Anforderungen sorgen direkt und indirekt für ein besseres Benutzererlebnis, eine einfachere Fehlersuche und -beseitigung durch den Entwickler sowie eine schnellere Einarbeitung für projektfremde Entwickler. Sie setzen somit ebenfalls ein Mindestmaß an Qualität voraus, bevor die Software dem Auftraggeber übergeben werden kann.

# 3 Technologie

Gemäß der funktionalen Anforderung FA-01 stellt der BPMN-Generator eine Benutzerschnittstelle über einen Webbrowser zur Verfügung. Demnach erfolgen clientseitige Benutzereingaben, die serverseitig der Anwendungslogik folgend verarbeitet werden.

Der Webservice ist mit dem Express.js-Framework in Node.js implementiert und persistiert die Daten durch ein MongoDB-Datenbankmanagementsystem (DBMS). Hieraus ergeben sich folgende Systemanforderungen:

- Node Version v12.18.3
- Express.js
- Mocha 8.1.3
- MongoDB Version 4.4.0

Konkrete Hintergründe zu den einzelnen Technologien werden in den folgenden Abschnitten vorgestellt.

## 3.1 Implementierungssprache: JavaScript und Node.js

Eine der prominentesten Programmiersprachen im Bereich der Webentwicklung ist JavaScript, welche ihren Ursprung in der Frontend-Entwicklung hat. Sie wurde in der ersten Hälfte der 1990er Jahre von Netscape für den Browser 'Netscape Navigator' zum Zweck, das Internet dynamischer zu machen, entwickelt [9]. Darauf folgte die Standardisierung der Sprache von der Normungsorganisation 'ECMA International' unter dem Namen ECMAScript [9].

Mit Node.js ist 2009 [9] zudem eine JavaScript-Laufzeitumgebung für das Backend hinzugekommen. Heutzutage stellt die Node.js-Entwicklung einen sehr prominenten Ansatz, moderne Webservices zu realisieren, dar. Die Laufzeitumgebung gestattet, sowohl im Frontend als auch im Backend JavaScript zu verwenden. Folglich werden Kontextwechsel reduziert, die Kommunikation vereinfacht sowie die Wiederverwendbarkeit des Codes erleichtert. Da es sich um eine plattformübergreifende Technologie handelt, lässt sich die Anwendung in vielen Ungebungen problemlos ausführen. Die Entwicklung betriebssystemabhängiger Clients ist ebenso nicht notwendig. Darüber hinaus stellen eine Vielzahl von Modulen Funktionen bereit, die den Entwicklungsaufwand reduzieren.

### 3.1.1 Framework: Express.js

Express.js bietet als Framework serverseitige Funktionalitäten zur Entwicklung von Webanwendungen. Es lässt sich leicht eine REST-Schnittstelle entwickeln, über die clientseitig Dienste aufgerufen werden können. Dies ermöglicht die Entwicklung unterschiedlicher Client-Implementationen, wohingegen die serverseitige Logik für alle Clients zur Verfügung steht. Für den BPMN-Generator zeigt sich hier ein großer Vorteil, wenn neben der Web-Benutzerschnittstelle zusätzlich beispielsweise eine Java-Applikation entwickelt werden soll. Ferner lässt sich das Routing in Express.js auf einfache Art und Weise konfigurieren und aufgrund der Unterstützung von View-Engines können Webpages mit dynamischen Inhalten gefüllt werden. Im BPMN-Generator Projekt kommt die View-Engine 'Pug' zum Einsatz.

### 3.1.2 Tests: Mocha

Zur Qualitätssicherung der Software ist das Schreiben und Ausführen von Tests unabdingbar. Tests können Implementierungsfehler frühzeitig aufdecken bzw. nach Codeänderungen die Korrektheit der Software indizieren. Durch sogenannte Black-Box-Tests wird die korrekte Funktionsweise von Softwareeinheiten im Hinblick auf ihre Ein- und Ausgabe geprüft. Meist werden als Eingabeparameter Repräsentanten verschiedener Werte-Äquivalenzklassen gewählt und es werden Grenzwerte geprüft. Mit einem Test-Framework kann dabei die notwendige Testumgebung aufgebaut werden und die Tests können kontrolliert und ggf. auch automatisiert ausge-

führt werden. In der Node.js-Umgebung unterstützt beispielsweise das Framework Mocha das Definieren und die Ausführung von Tests. Mocha ist als Entwicklungs-Abhängigkeit in der `package.json`-Datei gelistet.

## 3.2 Datenhaltung: MongoDB

Für die Persistierung der Daten dient ein MongoDB-DBMS. Die dokumentorientierte NoSQL-Datenbank ist zur Speicherung von Dokumenten im BSON-Format ausgelegt. Im Gegensatz zu einer relationalen Datenbank bietet die MongoDB mit den Dokumenten eine flexible Lösung, hierarchisch organisierte Daten in einem Eintrag zu speichern, ohne dabei ein Schema zu erzwingen [3]. Hierdurch ist es möglich, ohne einen großen Transformationsaufwand JSON-Objekte aus der Node.js-Anwendung in die Datenbank zu schreiben, bzw. von ihr auszulesen. Im BPMN-Generator Projekt dient das MongoDB-DBMS zur Persistierung der Diagramme (`Diagram`, siehe Abschnitt 5.3) und Diagramm-Sammlungen (`DiagramSet`, siehe Abschnitt 5.3).

# 4 Entwurf

Die Entwurfsphase ist für die Qualität und Wartbarkeit der Software von elementarer Bedeutung. Sie folgt auf die Analysephase und geht der Implementation voraus. Die Entscheidungen und Ergebnisse dieser Phase bestimmen die spätere Architektur, Struktur und Implementierung der Software. Die folgenden Abschnitte arbeiten die Entwurfsentscheidungen des BPMN-Generators heraus.

## 4.1 Systemstrukturierung

Die Struktur der BPMN-Generator Anwendung folgt dem Client-Server Modell. Der Server implementiert die gesamte Logik der Anwendung, während der Browser als Client nur die Benutzerschnittstelle anbietet. Zwischen Client und Server liegt ein Netzwerk, über welches sie kommunizieren können. Da lediglich ein Server die gesamte Funktionalität der Anwendung bereitstellt, lässt sich von einem zweischichtigen Client-Server-Modell reden. Ferner lässt sich der Client als thin-Client bezeichnen, da dieser keine Logik implementiert.

Ein entscheidender Vorteil des Client-Server-Entwurfs ist die zentrale Steuerung der Anwendung. Clients können von unterschiedlichen Orten auf den Dienst zugreifen und Daten verändern. Durch die Implementierung einer geeigneten API ist es darüber hinaus möglich, den Dienst plattformübergreifend und durch unterschiedliche Client-Implementationen zugänglich zu machen. Im Fall des BPMN-Generators genügt lediglich ein Browser, um die Anwendung zu nutzen.

Abbildung 4.1 zeigt den Kommunikationsablauf einer Anfrage des Clients als UML-Sequenzdiagramm. Ein HTTP-Request trifft bei der Anwendung ein, sofern notwendig werden Daten aus der MongoDB-Instanz abgefragt und schließlich sendet

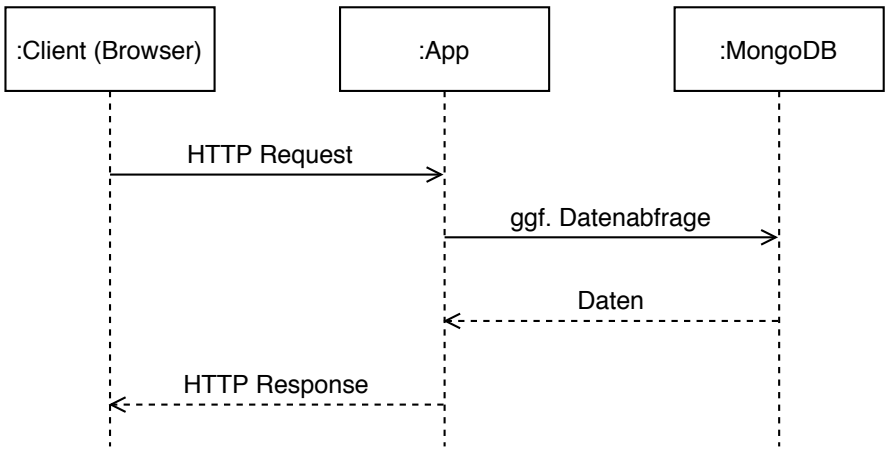


Abbildung 4.1: Das UML-Sequenzdiagramm stellt die Kommunikation der Anwendungskomponenten bei einem HTTP Request dar.

die Anwendung das Ergebnis der Anfrage zurück an den Client. Anfragen des Clients können demzufolge manipulierend auf Anwendungsdaten zugreifen. Anhand der Abbildung wird nochmals hervorgehoben, dass der Client lediglich die HTTP-Anfragen und -Antworten behandeln können muss, um die Dienste des Servers nutzen zu können.

## 4.2 Architekturentwurf

Der Architekturentwurf einer Anwendung ist eng verknüpft mit ihrer Wartbarkeit und Erweiterbarkeit. Wie auch schon im vorherigen Abschnitt, lassen sich hierbei grob die Komponenten ’MongoDB’, ’Client/Browser’ und ’App’ unterscheiden. Abbildung 4.2 zeigt diese Komponenten und ihre Verknüpfungen untereinander als UML-Komponentendiagramm. Die App implementiert die Logik und ist zwischen Datenspeicher und Client geschaltet. Ferner lässt sich die Anwendung selbst in weitere Teil-Komponenten untergliedern. Für die Kommunikation mit der Datenhaltungsschicht ist demnach eine Middleware zuständig, die von der Express.js-Komponente genutzt wird. Letztere definiert die Schnittstelle des Servers und bestimmt somit das Verhalten der Anwendung. Die konkrete Logik wiederum ist in weitere Komponenten ausgelagert, beispielsweise ist der BPMNGraphGenerator für die Generierung der BPMN-Diagramme zuständig.

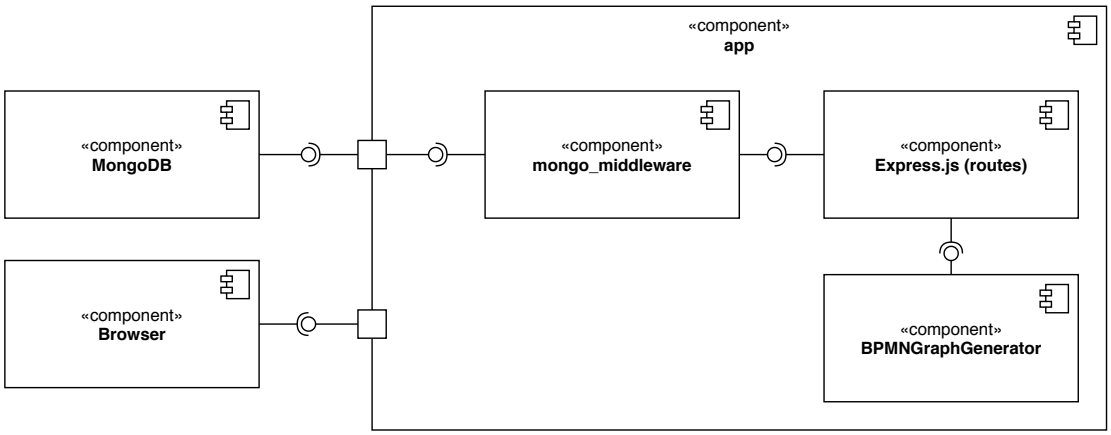


Abbildung 4.2: UML-Komponentendiagramm der wichtigsten Komponenten des BPMN-Generators.

Dieser Architekturentwurf ermöglicht durch die lose Kopplung zwischen Logik, Benutzerschnittstelle und Datenhaltung eine leichte Wartbarkeit. Durch die Middleware kann darüber hinaus Funktionalität zur Datenabfrage hinzugefügt werden oder sogar das gesamte DBMS ausgetauscht werden, ohne dass große Teile der Anwendung angepasst werden müssen. Es bedarf lediglich einer Anpassung der Middleware. Durch die Zentrierung der Logik in der serverseitigen Anwendung ist es darüber hinaus möglich, weitere Client-Implementationen bereitzustellen, die mit der Schnittstelle der Anwendung kommunizieren. Außerdem vereinfacht die Modularisierung der logischen Komponenten die Wiederverwendung des Codes in unterschiedlichen Programmteilen oder Projekten.

## 4.3 Feinentwurf: Klassen- und Objektentwurf

Ausführungsstartpunkt der Anwendung ist die Datei `/bin/www.js`, in welcher der HTTP-Server sowie die App (`app.js`) instanziiert werden. In der Datei `app.js` wird die Express.js-Anwendung erzeugt und konfiguriert. Insbesondere wird der von der Datei `routes/index.js` exportierte Router als Express.js-Middlewarefunktion für den Root-Pfad `/` gesetzt. Die bisher beschriebenen Programmteile setzen folglich das Fundament der Webservice-Anwendung und das Verhalten des Servers wird demnach durch die Konfiguration des Routers in der Datei `routes/index.js` bestimmt. Abbildung 4.3 zeigt die genannten und weitere Einheiten des BPMN-

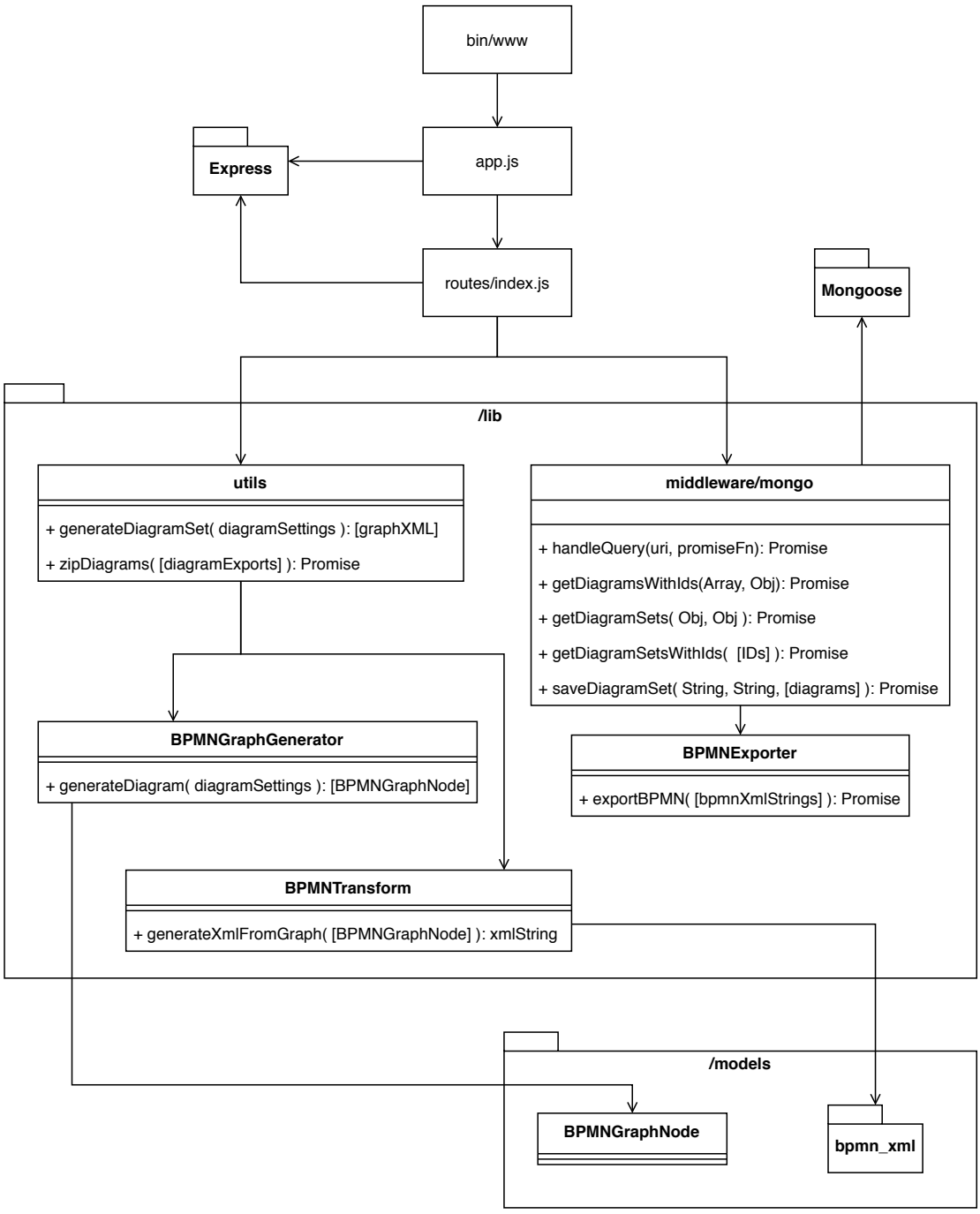


Abbildung 4.3: UML-Klassendiagramm des BPMN-Generators mit den zentralen Klassen und Paketen.

Generators. Auf die die Logik implementierenden Komponenten wird im Folgenden näher eingegangen.

Listing 4.1 zeigt zunächst die von der Utils-Datei zur Verfügung gestellten Funktionen. Zum einen generiert `generateDiagramSet(diagramSettings)` mehrere BPMN-Diagramme auf Grundlage der übergebenen Parameter. Hierfür werden zunächst durch den `BPMNGraphGenerator` Graphen erstellt, die die Struktur der generierten BPMN-Diagramme repräsentieren. Anschließend werden diese Graphen durch `BPMNTransform` in einen XML-String umgewandelt, um die standardisierte BPMN 2.0 Repräsentation des Diagramms zu erhalten. Zum anderen erstellt die Funktion `zipDiagrams(data)` eine ZIP-Datei aus allen übergebenen PDF-, PNG-, XML- und SVG-Dateien. Diese wird bei Erfolg als Buffer zurückgegeben.

```
1 module.exports = {
2     generateDiagramSet: generateDiagramSet,
3     zipDiagrams: zipDiagrams
4 };
```

Listing 4.1: Modulschnittstelle der utils-Datei.

Eine weitere zentrale Komponente der Anwendung ist die Middleware für das DBMS. Sie bietet Funktionen zur Abfrage und zum Schreiben von Daten der MongoDB-Datenbank an und nutzt für den Zugriff auf die Datenbank die Bibliothek 'Mongoose'. Außerdem bietet der `BPMNExporter` der Middleware die Funktionalität, BPMN-XML-Strings in weitere Austauschformate zu exportieren. Die Implementierung einer solchen Middleware bietet hinsichtlich des Entwurfs einige qualitative Vorteile. Einerseits können komplexere Zugriffsoperationen auf die Datenbank in einer Funktion gekapselt werden. Dadurch kann Code-Redundanz vermindert werden und die konkrete Implementation von der Funktion getrennt werden. Andererseits trennt die Middleware die Datenhaltungsschicht von der Logik und ermöglicht dadurch unkomplizierte Anpassungen der Datenzugriffe bei beispielsweise Versionswechseln der Datenbank oder einem Wechsel des gesamten DBMS.

Insgesamt ermöglicht der modularisierte Entwurf eine schnelle und einfache Anpassbarkeit einzelner Softwarekomponenten, eine leichtere Wiederverwendung implementierter Funktionen sowie eine bessere Einarbeitung von projektfremden Entwicklern, da die Komponenten eindeutiger identifiziert werden können.

# 5 Implementierung

Nach dem Entwurf steht im Zentrum des Softwareentwicklungsprozesses die Implementierung der Software. Auch wenn viele qualitätsbeeinflussende Entscheidungen bisher schon getroffen wurden, kommt es bei der Implementierung darauf an, den Code in einer nachvollziehbaren und verständlichen Art und Weise zu schreiben, zu gliedern und zu dokumentieren. Gegenstand dieses Kapitels sind konkrete Implementationsbeispiele des BPMN-Generators.

## 5.1 Server-Schnittstelle

Die Anwendung ist gemäß des Client-Server-Modells aufgebaut. Der Client sendet folglich Anfragen an den Server, welche dieser bearbeitet und beantwortet. Die Schnittstelle zwischen Client und Server lässt sich grundsätzlich als REST-API betrachten. Die Funktionen, die über die entsprechenden Pfade zur Verfügung gestellt werden, sind im Folgenden vorgestellt.

**GET**

| **Pfad** | **Funktion** |
|---|---|
| `/` | Leitet auf die Seite `/generate` weiter. |
| `/generate` | Erzeugt die 'Generator'-Ansicht. In dieser Ansicht wird dem Benutzer im Browser ein Formular für das Setzen der Diagramm-Parameter angezeigt. |

Tabelle 5.1: Client-Server Schnittstelle für HTTP-GET-Anfragen.

| Pfad | Funktion |
|---|---|
| `/my_models` | Erzeugt die 'Meine Diagramme'-Ansicht. Es werden alle in der Datenbank gespeicherten Diagramme geladen und übersichtlich im Browser dargestellt. |
| `/diagram/:diagramId/preview` | Zeigt die Vorschau-Ansicht des Diagramms mit der im Pfad definierten `diagramId` an. |
| `/diagram/:diagramId/download` | Lädt die Export-Dateien des Diagramms mit der im Pfad definierten `diagramId`, komprimiert diese in einer ZIP-Datei und sendet diese an den Client. |
| `/diagramset/:diagramSetId/preview` | Zeigt die Vorschau-Ansicht des `DiagramSets` mit der im Pfad definierten `diagramSetId` an. |
| `/diagramset/:diagramSetId/download` | Lädt die Export-Dateien der Diagramme aus der Diagramm-Sammlung mit der im Pfad definierten `diagramSetId`, komprimiert diese in einer ZIP-Datei und sendet diese an den Client. |

Tabelle 5.1: Client-Server Schnittstelle für HTTP-GET-Anfragen.

**POST**

| Pfad | Funktion |
|---|---|
| `/generate` | Generiert Diagramme auf Grundlage der mitgegebenen Eigenschaften. Leitet schließlich auf die Seite `/my_models` um. |

Tabelle 5.2: Client-Server Schnittstelle für HTTP-POST-Anfragen.

**DELETE**

| Pfad | Funktion |
|---|---|
| `/diagram/:diagramId` | Löscht das `Diagram`-Dokument mit der im Pfad definierten `diagramId`. |
| `/diagramset/:diagramSetId` | Löscht das `DiagramSet`-Dokument mit der im Pfad definierten `diagramSetId` inklusiver aller zugehörigen `Diagram`-Dokumente. |

Tabelle 5.3: Client-Server Schnittstelle für HTTP-DELETE-Anfragen.

Im aktuellen Entwurf ist diese Schnittstelle für die Ansteuerung mit einem Webbrowser ausgelegt. Durch entsprechende Anpassungen könnte diese jedoch auch von weiteren zu implementierenden Clients aufgerufen werden. Denkbar wäre auch ein Microservice, der die Dienste des BPMN-Generators nutzt, um in Folge von gewissen Ereignissen automatisiert Diagramme zu generieren. Somit ließe sich die Anwendung von der klassischen Mensch-Computer-Interaktion entkoppeln.

## 5.2 Generator

In Abschnitt 4.3 wurde bereits herausgearbeitet, dass zur Generierung der BPMN Diagramme die beiden Komponenten `BPMNGraphGenerator` und `BPMNTransform` zuständig sind. Erstere bietet die Funktion `generateDiagram(diagramSettings)` an, die ein Array von `BPMNGraphNodes` zurückliefert. Ein `BPMNGraphNode` ist eine einfache Datenstruktur, die einen Knoten in einem Graphen mit gewissen Vorgängern und Nachfolgern repräsentiert. Das zurückgegebene Array enthält alle Knoten, die ein Start-Event des generierten Diagramms repräsentieren. Darüber hinaus gibt das Funktionsargument `diagramSettings` die vom Benutzer eingegebenen Parameter zur Generierung der Diagramme an. Listing 5.1 zeigt ein Beispiel für das `diagramSettings`-Argument.

```
{
    noTasks: 7,
    noStartEvents: 2,
    noEndEvents: 3,
    noIntermediateEvents: 2,
    noXorSplits: 1,
    noXorJoins: 1,
    noAndSplits: 3,
    noAndJoins: 2,
    copies: 4
}
```

Listing 5.1: Ein Beispiel für das Konfigurations-Objekt diagramSettings.

Die andererseits von `BPMNTransform` bereitgestellte Funktion `generateXMLFromGraph` transformiert die Graph-Repräsentation des Diagramms in eine XML-Repräsentation, die anschließend gespeichert und in andere Formate exportiert werden kann. Die nächsten Abschnitte betrachten nochmals genauer den Weg von der Frontend-Benutzereingabe über die Generierung einer Graph-Repräsentation durch `generateDiagram` hin zur Umwandlung in eine XML-Datei durch die Funktion `generateXMLFromGraph`.

### 5.2.1 Benutzereingabe im Frontend

Zur Generierung der dynamischen Webpages dient die View-Engine 'Pug'. Für diese lassen sich sogenannte Templates erstellen, denen im Backend weitere Daten übergeben werden können. Das Rendern der Seite wird in den Routing-Funktionen des Express.js-Servers angestoßen, wie Listing 5.2 zeigt. Demzufolge wird bei einer HTTP-GET Anfrage an den Pfad `/generator` die Seite 'generator' mit dem Titel 'Generator' zurückgesendet. Diese wird von dem Webbrowser empfangen und dem Benutzer präsentiert.

In der Datei `/views/generator.pug` wird gemäß der Template-Sprache die Struktur der Seite definiert. Ein Auszug dessen findet sich in Listing 5.3. Die Datei definiert ein HTML-Formular, das für jeden Eingabeparameter ein `input`-Element hat.

Hierbei ermöglicht Pug die komfortable Möglichkeit, die `input`-Elemente innerhalb einer Schleife zu definieren, wodurch Redundanz reduziert werden kann. Abbildung 5.1 zeigt das so erzeugte Web-Formular in der Darstellung eines Webbrowsers.

```
1 router.get('/generator', (req, res) => {
2     res.render('generator', {title: 'Generator'});
3 });
```

Listing 5.2: Verhalten der Anwendung bei einer HTTP-GET-Anfrage auf dem Pfad '/generate'

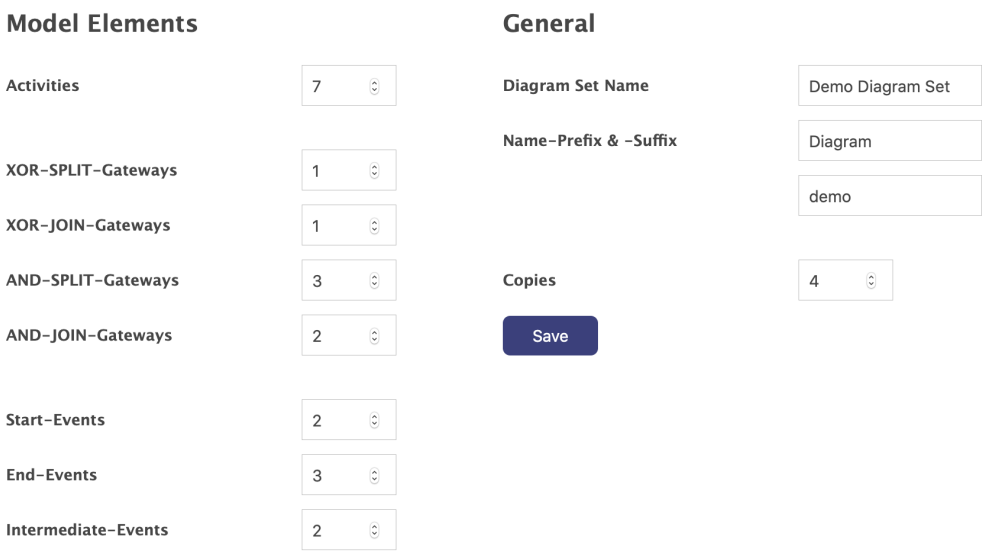


Abbildung 5.1: Die Browser-Darstellung der Generator-Seite.

Die Eingaben des Benutzers werden gemäß des `onchange`-Attributs von der Funktion `validateForm` im Frontend validiert und bei fehlerhaften Konfigurationen werden die betroffenen Eingaben visuell hervorgehoben. Sind alle Eingaben korrekt, wird die 'Save'-Schaltfläche aktiviert und beim Absenden durch den Benutzer wird gemäß der `form`-Definition eine HTTP-POST Anfrage an den Pfad `/generate` gesendet. Im Server wird daraufhin die Formular-Eingabe als `diagramSettings`-Objekt (siehe Listing 5.1) formatiert und der `generateDiagramSet`-Funktion der `utils`-Datei als Parameter übergeben. Listing 5.4 zeigt die Implementation dieser Funktion. Sie stellt das Bindeglied zwischen den in den nächsten beiden Abschnitten vorgestellten Funktionen `generateDiagram` und `generateXMLFromGraph` dar.

```
1  h1 Generator
2  form(action='/generate' method='post')
3      div(class='twocolumns generatorcolumns')
4          div.leftcolumn
5              h2 Model Elements
6              each input in ['Activities', 'XOR-SPLIT-
                   Gateways', 'XOR-JOIN-Gateways', 'AND-
                   SPLIT-Gateways', 'AND-JOIN-Gateways',
                   'Start-Events', 'End-Events',
                   'Intermediate-Events']
7                  div.row
8                      label(for= input)= input
9                      input(type='number' value=0 name=
                           input id= input, onchange=
                           'validateForm()')
10         div.rightcolumn
11             h2 General
12             div.row
13                 label(for='prefix_input') Diagram Set
                       Name
14                 input(type='text' name='diagram_set_name
                       ' id='diagramset_input' value='
                       Diagram Set')
15             // [...]
16             div.row
17                 button Save
```

Listing 5.3: Die Pug-Template Definition der Generator-Webpage in der Datei /views/generator.pug.

```
1  function generateDiagramSet (diagramSettings) {
2      let res = [];
3      for (let i = 0; i < diagramSettings.copies; i++) {
4          let bpmn_graph_heads = BPMNGraphGenerator.
               generateDiagram(diagramSettings);
```

```
5           let xml = BPMNTransform.generateXMLFromGraph(
                bpmn_graph_heads);
6           res.push(xml);
7       }
8       return res;
9   }
```

Listing 5.4: Die Implementation der Funktion 'generateDiagramSet'.

### 5.2.2 Generierung der Graph-Repräsentation durch 'generateDiagram'

Das Herzstück der Anwendung ist zweifelsfrei die Generierungslogik der Diagramme. Vereinfachend werden hierfür einige Einschränkungen vorgenommen, aus denen Nebenbedingungen für die Diagramme folgen. Der Generierungsprozess wird exemplarisch mit den in Listing 5.1 definierten Parameter-Werten vorgestellt, die, wie im vorherigen Abschnitt gezeigt, durch den Benutzer gesetzt werden.

```
1   function generateDiagram (diagramSettings) {
2       let remainingElements = Object.assign({},
            diagramSettings); //clone
3       let graphHeads = generateDiagramFramework(
            remainingElements);
4       enrichDiagramFramework(graphHeads, remainingElements
            );
5       return graphHeads;
6   }
```

Listing 5.5: Die Implementierung der Funktion 'generateDiagram'.

Der Startpunkt der Generierung ist die Funktion `generateDiagram` des `BPMNGraph-Generators`, deren Implementierung in Listing 5.5 zu sehen ist (weitere Auszüge des BPMNGraphGenerators lassen sich im Anhang nachschlagen). Zunächst wird durch die `generateDiagramFramework`-Funktion ein Grundgerüst des Diagramms

aufgebaut, in dem die Start- und End-Events sowie die Gateways enthalten sind, jedoch noch nicht alle Aktivitäten. Dieser Schritt hat bei gleichbleibender Eingabe stets das gleiche Ergebnis und gliedert sich in die `Join-Phase`, `Middle-Phase` und `Split-Phase`. Abbildung 5.2 zeigt die Ergebnisse dieser Phasen auf Basis der Eingabeparameter (siehe Listing 5.1) in grafischer Repräsentation.

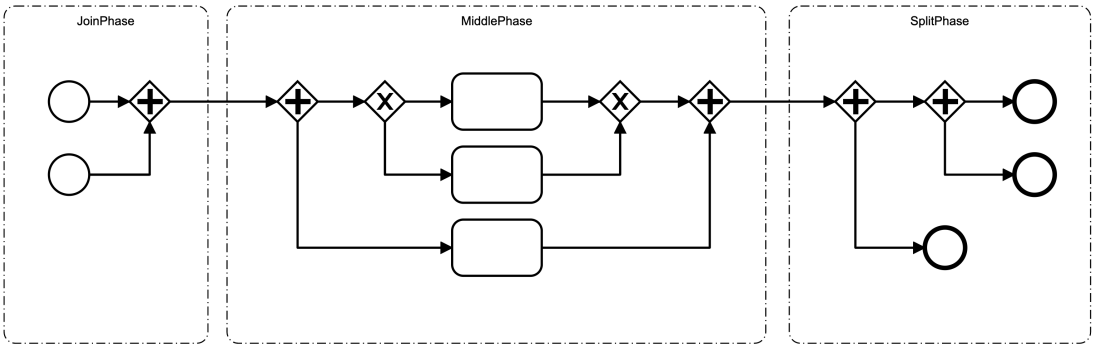


Abbildung 5.2: Die drei Generierungs-Phasen 'Join-Phase', 'Middle-Phase' und 'Split-Phase'

In der `Join-Phase` werden alle Start-Events zunächst durch AND-JOIN-Gateways zu einem Pfad zusammengeführt. Konkret werden die ersten beiden Events durch ein Gateway zusammengeführt und anschließend wird iterativ dieser neu entstandene Pfad durch ein weiteres Gateway mit dem nächsten Start-Event verbunden. Folglich werden in dieser Phase bei `n` Start-Events genau `(n-1)` AND-JOIN-Gateways benötigt. In Abbildung 5.2 werden lediglich zwei Start-Events mit einem Gateway zu einem Pfad zusammengeführt.

Sehr ähnlich zu der `Join-Phase` ist die `Split-Phase`, die umgekehrt arbeitet: Mit AND-SPLIT-Gateways wird der Pfad aufgespaltet, sodass jeder Pfad schließlich durch ein End-Event beendet wird. Bei `m` End-Events werden also `(m-1)` AND-SPLIT-Gateways benötigt. Im Beispiel sind es zwei Gateways, die in drei End-Events münden.

Schließlich werden in der Middle-Phase alle verbliebenen Gateways positioniert. Diese werden dabei geschachtelt und jeweils mit einer Aktivität verbunden. Die Zahl der (verbliebenen) AND-SPLIT- bzw. XOR-SPLIT-Gateways muss also mit der Zahl der AND-JOIN- bzw. XOR-JOIN-Gateways übereinstimmen. Bei `l` SPLIT- und `l` JOIN-Gateways muss es für die Middle-Phase mindestens `(l+1)` Aktivitäten geben.

Aus den in der `Join-`, `Middle-` und `Split-Phase` getroffenen Modellierungsregeln ergeben sich die im Folgenden gelisteten Vorbedingungen für die Diagramme:

$$\#StartEvents > 0$$
$$\#EndEvents > 0$$
$$\#XOR\text{-}JOIN \geq 0$$
$$\#XOR\text{-}JOIN = \#XOR\text{-}SPLIT$$
$$\#AND\text{-}JOIN \geq (\#StartEvents - 1)$$
$$\#AND\text{-}SPLIT \geq (\#EndEvents - 1)$$
$$\#AND\text{-}JOIN - (\#StartEvents - 1) = \#AND\text{-}SPLIT - (\#EndEvents - 1)$$
$$\#AND\text{-}JOIN - (\#StartEvents - 1) + \#XOR\text{-}JOIN + 1 \leq \#Tasks$$

Diese Bedingungen werden bei der Programmausführung durch die in Listing 5.6 angegebene Funktion `validateDiagramSettings` geprüft. Sollten diese nicht erfüllt sein, wird ein Fehler geworfen und der Benutzer wird über eine Ausgabe informiert. Darüber hinaus werden diese Bedingungen ebenfalls im Frontend geprüft und ungültige Eingaben werden nicht zugelassen.

```
1  function validateDiagramSettings(diagramSettings) {
2      // at least one start- and one end-event
3      let valid = diagramSettings.noStartEvents > 0 &&
           diagramSettings.noEndEvents > 0;
4
5      // correct number of Gateways in join- and split-
           phase
6      valid = valid && diagramSettings.noAndJoins >= (
           diagramSettings.noStartEvents - 1)
7          && diagramSettings.noAndSplits >= (
               diagramSettings.noEndEvents - 1)
8
9      // correct number of Gateways in middle-phase
10     valid = valid && diagramSettings.noXorJoins >= 0
11         && diagramSettings.noXorJoins ===
```

```
            diagramSettings.noXorSplits
12          && (diagramSettings.noAndJoins - (
            diagramSettings.noStartEvents - 1)
13           === diagramSettings.noAndSplits - (
               diagramSettings.noEndEvents - 1))
14
15      // enough tasks
16      valid = valid && diagramSettings.noAndJoins -
           diagramSettings.noStartEvents
17           + diagramSettings.noXorJoins + 1 <=
              diagramSettings.noTasks
18
19      if (!valid) {
20          throw new Error('Invalid diagram Settings.');
21      }
22  }
```

Listing 5.6: Implementierung der Funktion validateDiagramSettings.

Nachdem nun das Grundgerüst des Diagramms als Graph-Repräsentation generiert wurde, übernimmt die Funktion `enrichDiagramFramework`. Diese fügt die verbliebenen Aktivitäten und Intermediate-Events zufällig in das Diagramm ein. Das auf Grundlage der Benutzereingabe (Listing 5.1) generierte Grundgerüst (Abbildung 5.2) wird folglich angereichert und nimmt beispielsweise die in Abbildung 5.3 vorliegende Gestalt an. Durch die Wiederholung dieses Prozesses lassen sich schließlich mehrere Diagramme generieren. Im folgenden Abschnitt wird betrachtet, wie die auf diesem Wege generierten Diagramm-Graphen in ein XML-Format umgewandelt werden.

### 5.2.3 Transformierung in XML-Repräsentation durch 'generateXMLFromGraph'

BPMN 2.0 stellt einen Modellierungsstandard für Geschäftsprozesse dar. Teil dieses Standards ist die Definition der Syntax und Semantik aller grafischen Modellierungselemente. Darüber hinaus definiert BPMN 2.0 eine maschinell verarbeit-

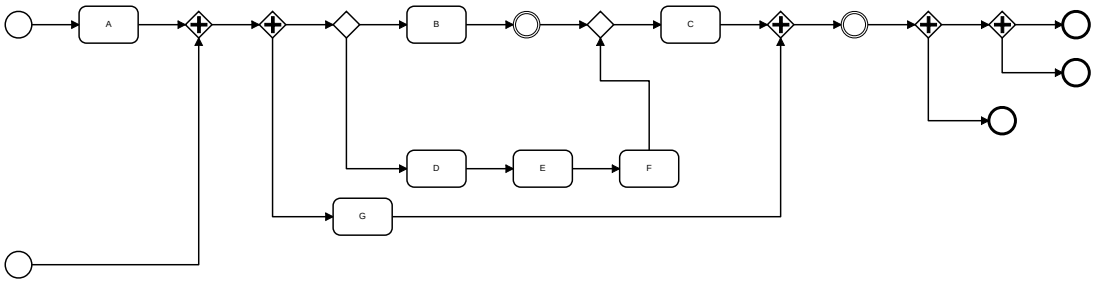


Abbildung 5.3: Ein Beispiel für ein generiertes Diagramm, das aus dem Grundgerüst in Abbildung 5.2 angereichert wurde, indem enrichDiagramFramework die verbliebenen Aktivitäten zufällig eingefügt hat.

bare Repräsentation im XML-Format [4]. Zur Transformation eines BPMN-Modells aus der Graph-Repräsentation in ein XML-Dokument ist demnach die Kenntnis des XML-Schemas notwendig.

Für die Verarbeitung durch das Programm bietet es sich an, eine Klassen-Repräsentation der einzelnen XML-Elemente zu definieren. Diese Klassen können für jedes Element neu instanziiert werden und ihre Eigenschaften können einfach angepasst werden. Außerdem ermöglicht die Überführung in die Klassen-Repräsentationen die Beachtung von Typ-Hierarchien der XML-Elemente, die in Abbildung 5.4 dargestellt sind. So ist beispielsweise ein `FlowElement` der übergeordnete Typ aller im Diagramm darstellbaren Elemente. Diese gliedern sich auf in sogenannte `FlowNodes`, also alle Knoten des Diagramms, sowie die als `SequenceFlows` bezeichneten Verbindungspfeile. Durch die Zusammensetzung dieser XML-Objekte entsteht schließlich eine vollständige Repräsentation der zu generierenden XML-Datei, die anschließend in einen XML-String umgewandelt wird.

```
1 function generateXMLFromGraph(startNodes)
```

Listing 5.7: Signatur der Funktion generateXMLFromGraph.

Rekapitulierend ist das Ergebnis des in Abschnitt 5.2.2 vorgestellten Generierungsprozesses eine Graph-Repräsentation (mit `BPMNGraphNodes`) des Diagramms. Anstelle aller Knoten des Graphen genügt es, die Startknoten stellvertretend für den gesamten Graphen anzugeben, da die verbliebenen Knoten iterativ aus den Nachfolgern ermittelt werden können. Wie Listing 5.7 zeigt, werden ebendiese Startkno-

ten der Funktion `generateXmlFromGraph` als Parameter übergeben. Anschließend wird wie beschrieben die Struktur der XML-Repräsentation rekursiv durch Objekte aufgebaut und in einen XML-String umgewandelt.

## 5.3 Datenhaltung

Die Datenhaltung erfolgt in einem MongoDB-DBMS. Die URI des MongoDB-Servers lässt sich in der Datei `/app/settings/mongo_settings.json` konfigurieren. Zuständig für den Zugriff auf die Datenbank ist die Middleware des MongoDB-DBMS (`/lib/middleware/mongo.js`), die Funktionen zum Lesen und Schreiben von Daten anbietet. Auf dem MongoDB-Server sind die beiden Collections 'diagrams' und 'diagramSets' angelegt, in denen jeweils Dokumente abgelegt sind. Die korrespondierenden Schemas sind entsprechend mit `Diagram` und `DiagramSet` benannt.

Das `Diagram`-Schema (siehe Listing 5.8) definiert mehrere Felder, die neben den unterschiedlichen Austauschformaten eines Diagramms zusätzlich Metainformationen, wie das Erstellungsdatum oder den Namen des Diagramms speichern. Das Feld `diagramId` stellt eine eindeutige Kennung des Dateneintrags dar.

```
1  {
2      "diagramId": { "type": "String", "index": true },
3      "name": "String",
4      "createdAt": "Date",
5      "lastChangedAt": { "type": "Date", "default": "Date.
           now" },
6      "xml": "String",
7      "svg": "String",
8      "png": "Buffer",
9      "pdf": "Buffer"
10 }
```

Listing 5.8: MongoDB-Schema des Typs 'Diagram'.

Bei der Generierung von Diagrammen werden meist mehrere Diagramme generiert. Jedes einzelne wird gemäß des Schemas aus Listing 5.8 in der `diagrams`-Collection als einzelnes Dokument gespeichert. Um diese zusammengehörigen Diagramme gruppieren zu können, wird zusätzlich ein sogenanntes `DiagramSet` gespeichert. Dieses enthält neben Metainformationen eine Liste aller `diagramIds`, die zum entsprechenden `DiagramSet` gehören (siehe Listing 5.9).

```
{
    "diagramSetId": "String",
    "diagramSetName": "String",
    "createdAt": "Date",
    "diagramIds": ["String"]
}
```

Listing 5.9: MongoDB-Schema des Typs 'DiagramSet'.

### 5.3.1 Die handleQuery-Funktion

Die Middleware der MongoDB exportiert sämtliche Funktionen zur Abfrage der Datenbank. Eine zentrale Funktion ist allerdings die `handleQuery`-Funktion, die bei jeder Abfrage verwendet werden muss. Die Implementation dieser Funktion ist in Listing 5.10 gezeigt.

```
function handleQuery(uri, asyncFn) {
    return new Promise((resolve, reject) => {
        initializeConnection(uri).then(() => {
            asyncFn().then((data) => {
                closeConnection().then(() => {
                    resolve(data);
                }).catch(reject);
            }).catch((err) => {
                closeConnection().finally(() => {
                    reject(err);
                });
```

```
12              });
13          }).catch((err) => {
14              closeConnection().finally(() => {
15                  reject(err);
16              });
17          });
18      });
19  }
```

Listing 5.10: Die Implementation der Funktion 'handleQuery'.

Diese Funktion stellt eine generische Implementation der Datenbankabfrage dar. Grundsätzlich kann hierdurch sichergestellt werden, dass vor jeder Datenbankabfrage eine Verbindung zur Datenbank hergestellt wird und insbesondere, dass diese nach der Abfrage wieder geschlossen wird. Die eigentliche Abfrage der Daten ist in der als Parameter übergebenen Funktion `asyncFn` definiert, die ein `Promise` zurückliefern muss und nach erfolgreichem Verbindungsaufbau ausgeführt wird. Durch diesen Entwurf wird dem Entwickler die Verantwortung für den Verbindungsaufbau und -abbau abgenommen und die Entstehung inkonsistenter Zustände wird unterbunden. Darüber hinaus wird redundanter Code vermieden, der durch den wiederholten Aufbau und Abbau der Verbindung entstehen würde.

## 5.4 Datenexport

Zum Austausch der Diagramme sollen diese in unterschiedliche Formate exportiert werden können. Die Exporte werden bereits beim Speichern des Diagramms in die Datenbank geschrieben. Die Funktionalität des Exportierens wird von der Funktion `exportBPMN` des `BPMNExporters` bereitgestellt. Diese speichert zunächst temporäre Dateien der XML-Repräsentation im `.bpmn`-Format, welche anschließend durch das Fremdmodul `bpmn-to-image` in das SVG- und PNG-Format umgewandelt werden. Daraufhin wird die SVG-Datei in eine PDF-Datei exportiert und die temporären Dateien werden gelöscht. Diese Export-Dateien werden im Falle des XML- und SVG-Formats als UTF-8 codierter String in die Datenbank geschrieben. Das PNG- und PDF-Format wird in binärer Repräsentation gespeichert.

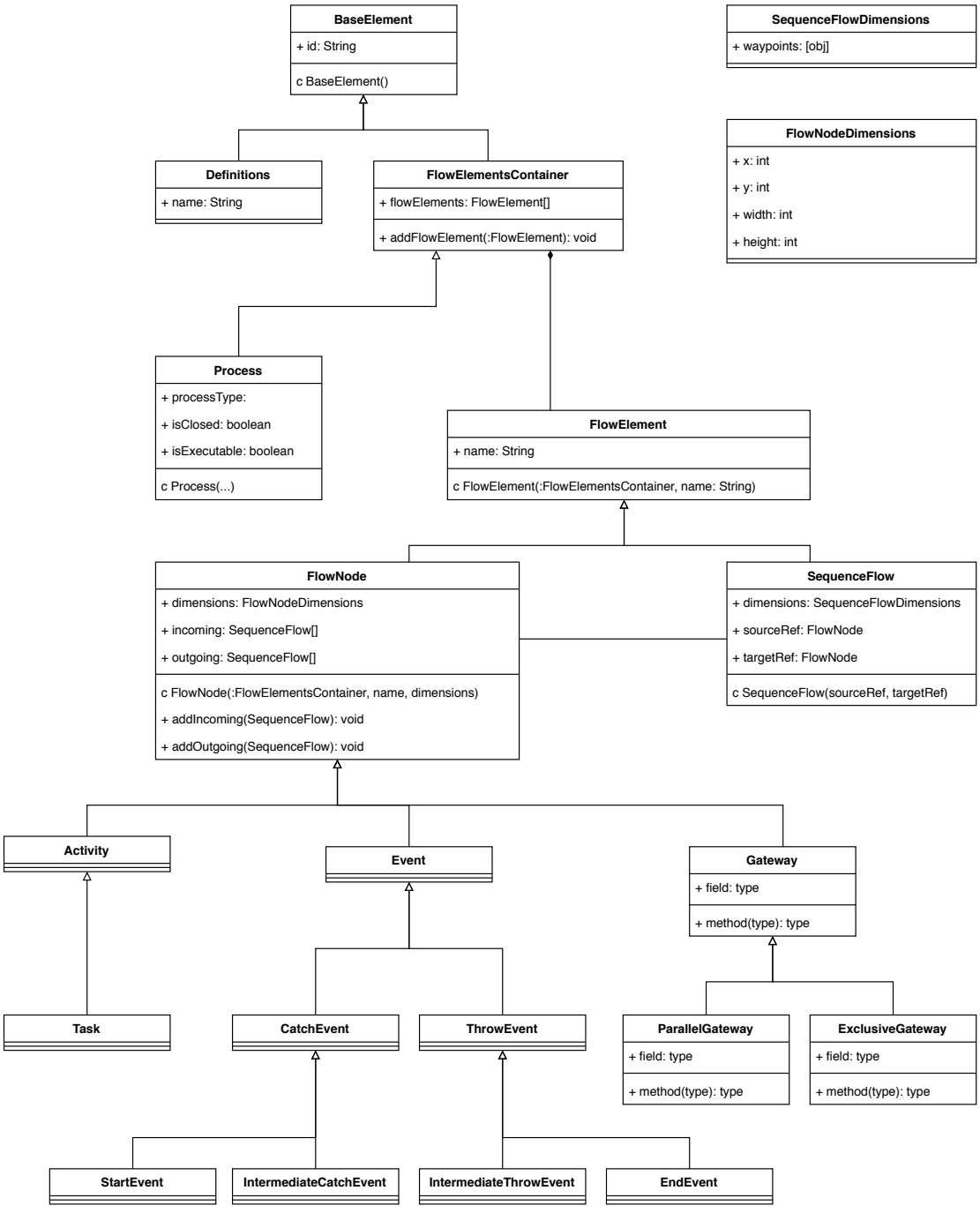


Abbildung 5.4: UML-Klassendiagramm der Klassen, welche die XML-Elemente auf der Grundlage des im BPMN 2.0 Standards definierten XML-Schemas repräsentieren.

# 6 Anforderungsabgleich

Nach der Fertigstellung der Implementation ist es sinnvoll, den Erfolg des Softwareentwicklungsprozesses zu beurteilen. Hierbei können die anfangs definierten Anforderungen an das System als Maßstab herangezogen werden. Dabei ist es möglich, dass alle Anforderungen vollständig erfüllt wurden oder sich diese im Laufe des Prozesses verändert haben und somit verfehlt wurden. Die folgende Tabelle gibt eine Übersicht des Anforderungsabgleichs für den BPMN-Generator.

| Name | +/o/- | Beschreibung |
|---|---|---|
| FA-01: Plattform | + | Der Express.js-Server stellt seinen Dienst über die durch einen Webbrowser erreichbare Benutzerschnittstelle bereit. |
| FA-02: Benutzerschnittstelle | + | Die Ansichten sind über die Pfade `/my_models` und `/generator` erreichbar. |
| FA-03: Eingabeparameter des Generators | | Die geforderten Eingabeparameter sind gegeben. Es wird zusätzlich zwischen SPLIT- und JOIN-Gateways unterschieden. Außerdem kann auch die Anzahl der Intermediate-Events gesetzt werden. |
| FA-04: Valide Syntax | o | Die generierten Diagramme können aufgrund der vorgenommenen Einschränkungen nur eine valide Syntax aufweisen, auch wenn hierdurch die Anzahl der möglichen generierbaren Diagramme gemindert wird. Sollten die eingegebenen Parameter keine entsprechende Generierung zulassen, wird dem Nutzer darüber hinaus eine Fehlermeldung ausgegeben. Es ist jedoch nicht garantiert, dass sich alle generierten Diagramme unterscheiden. |

| | | |
|---|---|---|
| FA-05: Persistenz | o | Im Laufe der Entwicklung stellte es sich als sinnvoller heraus, alle generierten Diagramme in der Datenbank zu speichern und diese anschließend über einen Datenexport verfügbar zu machen. Ein Vorteil dessen ist, dass die Anwendung dadurch als Verwaltungstool agieren kann und alle generierten Diagramme gesammelt zu finden sind. |
| FA-06: Ausgabeformat | + | Es ist ein Export in die Formate XML, PDF, PNG und SVG möglich. |
| NA-01: Wartbarkeit und Erweiterbarkeit | + | Die Modularisierung der Software wurde in dieser Ausarbeitung mehrmals angesprochen. Sowohl die Struktur des Systems als auch der konkrete Feinentwurf berücksichtigen den Grundsatz der Modularisierung. Hierdurch wird die Wartbarkeit und Erweiterbarkeit des Systems verbessert. |
| NA-02: Robustheit | + | Es erfolgt sowohl im Frontend als auch im Backend eine Validierung der Benutzereingaben. Bei fehlerhaften Eingaben wird der Benutzer durch entsprechende Hinweise informiert. |
| NA-03: Behandlung von Laufzeitfehlern | o | Laufzeitfehler, wie z.B. die Nichterreichbarkeit der MongoDB-Instanz, werden vom System angemessen behandelt und dem Benutzer wird dieser meist inklusive 'Stacktrace' ausgegeben. Aus diesem Grund wurde auf eine ausführlichere Dokumentation der Fehler in einer Log-Datei verzichtet, da der Anwender dieser Anwendung meist unmittelbar an der Behebung des Fehlers beteiligt ist. |
| NA-04: Testabdeckung | + | Es wurden Unit-Tests entwickelt, die die zentralen Funktionen der Anwendung testen. |
| NA-05: Dokumentation | + | Der Quellcode zeichnet sich durch eine ordentliche Struktur sowie einer Dokumentation der Funktionen aus. |

Tabelle 6.1: Anforderungsabgleich der funktionalen und nichtfunktionalen Anforderungen. Für die Bewertung dienen die Zeichen '+' = 'vollständig erfüllt', 'o' = 'teilweise erfüllt' und '-' = 'nicht erfüllt'.

# 7 Verwandte Arbeiten

Zur kontextuellen Abgrenzung und Hervorhebung von Vorteilen der entwickelten Anwendung 'BPMN-Generator' ist es sinnvoll, verwandte Softwarelösungen zu betrachten. Untersuchungsgegenstand sind die Modellierungstools 'Signavio Process Manager', 'diagrams.net', 'bpmn.io', 'Lucidchart' und 'Zeebe Modeler'.

## 7.1 Signavio Process Manager

Der Signavio Process Manager ist ein Tool zur Modellierung von BPMN 2.0 Prozessen. Gemeinsam mit anderen Tools kann es ferner zur Dokumentation, Analyse und Optimierung der Prozesse eingesetzt werden. Somit handelt es sich bei dem Signavio Process Manager um ein sehr professionelles aber auch kostenpflichtiges Tool zur Verwaltung von BPMN Prozessmodellen. [1]

## 7.2 diagrams.net

Diagrams.net (ehemals draw.io) ist eine Software zur Erstellung von Diagrammen. Neben UML- und Entity Relation Modellierungselementen stellt das Tool ebenfalls BPMN-Elemente zur Modellierung bereit. Die Anwendung kann über einen Browser oder auch über Desktop-Anwendungen aufgerufen werden. Die vom Nutzer erstellten Diagramme können in gängige Austauschformate exportiert werden, allerdings lässt sich keine BPMN 2.0-konforme XML-Datei erstellen. Insgesamt bietet das Tool sehr flexible Möglichkeiten, grafische Modelle manuell zu erstellen und zu bearbeiten. [5]

## 7.3 bpmn.io

Die Camunda Services GmbH stellt mit bpmn.io ein kostenfreies Tool zur Modellierung von BPMN 2.0 Prozessmodellen bereit. Das Tool ermöglicht das Erstellen und Bearbeiten von Modellen im Browser. Die Modelle können anschließend im XML- oder SVG-Format heruntergeladen werden. Demnach handelt es sich um ein übersichtliches und benutzerfreundliches Tool, das sich auch in eigene Anwendungen integrieren lässt. [10]

## 7.4 Lucidchart

Lucidchart ist eine kostenpflichtige Software für die professionelle Modellierung von BPMN-Diagrammen und vielen weiteren Modellierungstypen. Das Tool ist für die Zusammenarbeit im Team ausgelegt und ermöglicht somit eine feine Zugriffsverwaltung sowie Kommentar- und Kommunikationsmöglichkeiten innerhalb der Software. Außerdem bietet Lucidchart die Integration von Drittanbieterservices wie beispielsweise Atlassians Projektplanungssoftware Jira oder dem Kommunikationstool Slack an. Somit lässt sich die Software als benutzerfreundlich charakterisieren und bietet über die Modellierung von BPMN Diagrammen hinaus noch eine umfangreiche Kollaborationsumgebung an. [2]

## 7.5 Zeebe Modeler

Mit dem zu der Workflow-Engine Zeebe gehörenden 'Zeebe Modeler' lassen sich ebenfalls BPMN-Diagramme definieren. Zeebe ermöglicht die Orchestrierung von Microservices auf Grundlage eines in BPMN definierten Prozesses. Die Prozessdiagramme lassen sich im XML-Format oder als Bilddatei speichern. Doch im Gegensatz zu den bisher vorgestellten Modellierungsumgebungen dient der Zeebe Modeler nicht nur ausschließlich der visuellen Modellierung von Prozessen. Ferner können Schnittstellen zu automatisierten Aktivitäten und Events konfiguriert werden. Es können also technische Einstellungen vorgenommen werden, mit Hilfe derer die Workflow-Engine die Aktivitäten orchestrieren kann. Damit geht der

Umfang des Zeebe Modelers über die Grundfunktionalitäten eines grafischen Modellierungstools hinaus. [11]

## 7.6 Abgrenzung des BPMN-Generators

Die vorgestellten Modellierungstools bieten allesamt die Möglichkeit an, Geschäftsprozesse manuell gemäß BPMN 2.0 zu modellieren. Der im Rahmen dieser Arbeit entwickelte BPMN-Generator ermöglicht nun zusätzlich, BPMN-Modelle auf der Basis verschiedener Eingabeparameter automatisch zu generieren. Die bereitgestellten Ausgabeformate ermöglichen zudem beispielsweise die manuelle Bearbeitung eines generierten Diagramms in bpmn.io.

# 8 Zusammenfassung

Der Standard BPMN 2.0 ist im Bereich der Prozessmodellierung ein Gegenstand elementaren Interesses. Es existieren zahlreiche Modellierungstools, mit deren Hilfe sich BPMN-Diagramme manuell erstellen und in unterschiedliche Formate exportieren lassen. Damit die für Studienzwecke im Rahmen der Wahrnehmungs- und Verständlichtkeitsforschung von BPMN 2.0 Prozessmodellen benötigten Diagramme nicht stets händisch modelliert werden müssen, wurde im Rahmen dieser Arbeit der BPMN-Generator zur automatisierten Generierung von Prozessmodellen entwickelt.

Die in der Anforderungsanalyse erarbeiteten funktionalen und nichtfunktionalen Anforderungen an die Software stellen im Laufe des Softwareentwicklungsprozesses die Kommunikationsgrundlage und Zielvorgabe des Produkts dar. Es ist eine Webanwendung zu entwickeln, die auf Basis von Parametereingaben des Benutzers automatisiert syntaktisch korrekte BPMN-Diagramme erstellt und diese in gewissen Austauschformaten zum Export bereitstellt. Technologisch soll die Anwendung in einer Node.js-Umgebung mit dem Framework Express.js und dem MongoDB-DBMS implementiert werden.

Der Client-Server-Entwurf der Software zeichnet sich durch einen hohen Modularisierungsgrad und eine Entkopplung der einzelnen Komponenten sowie insbesondere der Datenhaltungs-, Benutzer- und Logikschicht aus. Dies verbessert die Wartbarkeit und Änderbarkeit des Systems.

Der abschließende Anforderungsabgleich hat gezeigt, dass eine Vielzahl der anfangs definierten Anforderungen an das System erfüllt werden konnten. Das Entwicklungsprojekt ist demnach zu einem erfolgreichen Abschluss gekommen.

## 8.1 Ausblick

Im Hinblick auf die Erweiterbarkeit des BPMN-Generators offenbaren sich viele mögliche Perspektiven. Zum einen kann die Generierung der BPMN-Diagramme verbessert werden, indem die Regelmenge erweitert und verfeinert wird. Außerdem kann der Umfang des unterstützten BPMN 2.0 Standards erweitert werden, sodass beispielsweise auch unterschiedliche Eventtypen sowie Pools und Swimlanes generiert werden können. Unter Umständen ist es ebenfalls vorteilhaft, bereits generierte Diagramme direkt in der Anwendung bearbeiten zu können, indem z.B. der bpmn.io-Editor integriert wird. Dies würde dem Benutzer auf schnelle und komfortable Weise das Ausbessern bzw. Anpassen der Diagramme ermöglichen. Schließlich ist auch eine Benutzer- und Zugriffsverwaltung denkbar, wodurch Benutzer sich authentifizieren müssen, um auf den Service zugreifen zu können.

Demzufolge zeigen sich mehrere Erweiterungsansätze, um die Grundfunktionalität der aktuellen Anwendung zu erweitern. Folglich zeichnet sich das Potenzial des BPMN-Generators ab, sich zu einer professionellen Modellierungs-, Generierungs- und Verwaltungsumgebung von BPMN-Prozessmodellen zu entwickeln.

# A Quelltexte

## A.1 BPMNGraphGenerator.js

*Hinweis: Dieses Listing enthält nur einen Auszug der Quelldatei.*

```
1  /**
2   * Generates random diagram-graph (BPMNGraphNode) and
       returns start-nodes. First a 'static' framework is
       created which
3   * is randomly enriched with remaining diagram-elements
4   * @param diagramSettings: {noStartEvents, noEndEvents,
       noIntermediateEvents, noTasks, noAndSplits,
5   * noAndJoins, noXorSplits, noXorJoins} is not modified
       by this function
6   * @return [BPMNGraphNode] Start-nodes of the generated
       diagram-graph
7   */
8  function generateDiagram (diagramSettings) {
9      let remainingElements = Object.assign({},
          diagramSettings); //clone
10     let graphHeads = generateDiagramFramework(
          remainingElements);
11     enrichDiagramFramework(graphHeads, remainingElements
          );
12     return graphHeads;
13 }
14
```

```
15 /**
16  * Generate simple Framework of the diagram containing
       Start- and End-Events and Gateways. Construction
       follows
17  * certain rules (Join-Phase, Middle-Phase, Split-Phase)
        and results in the same diagram-graph when
       diagramSettings
18  * stay the same.
19  * @param diagramSettings is modified by this function
20  * @returns [BPMNGraphNode] containing the start-nodes
       of the framework.
21  */
22 function generateDiagramFramework (diagramSettings) {
23     // validation
24     validateDiagramSettings(diagramSettings);
25
26     let {joinPhaseGateways, middlePhaseGateways,
          splitPhaseGateways} = getPhaseGateways(
          diagramSettings);
27
28     // handle Join-, Split- and Middle-Phase
29     let { heads, tail } = handleJoinPhase(
          joinPhaseGateways, diagramSettings);
30     let splitPhaseHead = handleSplitPhase(
          splitPhaseGateways, diagramSettings);
31     handleMiddlePhase(tail, splitPhaseHead,
          middlePhaseGateways, diagramSettings);
32
33     return heads;
34 }
35
36 /**
37  * Validates DiagramSettings considering the
       preconditions.
38  * @param diagramSettings settings of the diagram.
```

```
39  */
40 function validateDiagramSettings(diagramSettings) {
41     // correct number of Gateways
42     let valid = (diagramSettings.noAndJoins -
           diagramSettings.noStartEvents ===
43          diagramSettings.noAndSplits - diagramSettings.
               noEndEvents) &&
44          diagramSettings.noXorJoins === diagramSettings.
               noXorSplits;
45
46     // at least one start- and one end-event
47     valid = valid && diagramSettings.noStartEvents > 0
           && diagramSettings.noEndEvents > 0;
48
49     // enough tasks
50     valid = valid && diagramSettings.noAndJoins -
           diagramSettings.noStartEvents + diagramSettings.
           noXorJoins < diagramSettings.noTasks
51
52     if (!valid) {
53         throw new Error('Invalid diagram Settings.');
54     }
55 }
56
57 /**
58  * Takes diagram-graph with basic elements and adds
        remaining Tasks and Events randomly
59  * @param heads [BPMNGraphNode] Start-Nodes of the
        diagram-graph
60  * @param diagramSettings The settings of the diagram
61  */
62 function enrichDiagramFramework(heads, diagramSettings)
       {
63     let remainingElements = Object.assign({},
           diagramSettings); //clone
```

```
64      let diagramNodes = getDiagramNodes(heads);
65      // filter end events
66      diagramNodes = diagramNodes.filter((el) => el.type
           !== DiagramElementType.END_EVENT);
67
68      while (remainingElements.noTasks > 0 ||
           remainingElements.noIntermediateEvents > 0) {
69          let insertNode;
70          if (remainingElements.noTasks > 0) {
71              insertNode = new Node(DiagramElementType.
                   TASK);
72              remainingElements.noTasks--;
73          } else if (remainingElements.
               noIntermediateEvents > 0) {
74              insertNode = new Node(DiagramElementType.
                   INTERMEDIATE_EVENT);
75              remainingElements.noIntermediateEvents--;
76          }
77
78          // select random node and random successor of
               this node and insert new node between them
79          let randomIndex = Math.floor(Math.random() *
               diagramNodes.length);
80          let randomSuccessorIndex = Math.floor(Math.
               random() * diagramNodes[randomIndex].
               successors.length);
81          insertNodeBetween(insertNode,
82              diagramNodes[randomIndex],
83              diagramNodes[randomIndex].successors[
                   randomSuccessorIndex]);
84      }
85  }
```

# Literatur


[1] *BPMN-Suite zur Prozessmodellierung - Signavio Process Manager*. https://www.signavio.com/de/products/process-manager/. 28.10.2020.

[2] *BPMN Tool zur Prozessmodellierung*. https://www.lucidchart.com/pages/de/beispiele/bpmn-tool. 22.10.2020.

[3] S. Bradshaw und K. Chodorow. *MongoDB: The Definitive Guide, 3rd Edition*. O'Reilly Media, Incorporated, 2018. ISBN: 9781491954454. URL: https://books.google.de/books?id=VS-2tAEACAAJ.

[4] *Business Process Model and Notation (BPMN)*. Object Management Group (OMG). Jan. 2011.

[5] *Diagram Software and Flowchart Maker*. https://www.diagrams.net. 28.10.2020.

[6] M. Dumas, M. L. Rosa, J. Mendling und H. A. Reijers. *Fundamentals of Business Process Management*. Springer Berlin Heidelberg, 2013.

[7] Jakob Freund und Bernd Rücker. *Praxishandbuch BPMN*. 5. Aufl. Carl Hanser Verlag München, 2017.

[8] J Mendling, H. A. Reijers und W. M. P. van der Aalst. *Seven Process Modeling Guidelines (7PMG)*. Techn. Ber. Humboldt University und Eindhoven University of Technology, 2009.

[9] Axel Rauschmayer. *Speaking JavaScript*. 1st. O'Reilly Media, Inc., 2014. ISBN: 1449365035.

[10] *Web-based tooling for BPMN, DMN and CMMN*. https://bpmn.io. 28.10.2020.

[11] *Zeebe - Workflow Engine for Microservices Orchestration*. https://zeebe.io. 28.10.2020.

[12] Michael Zimoch, Rüdiger Pryss, Georg Layher, Heiko Neumann, Thomas Probst, Winfried Schlee und Manfred Reichert. „Utilizing the Capabilities Offered by Eye-Tracking to Foster Novices' Comprehension of Business Porcess Models“. In: *International Conference on Cognitive Computing (ICCC'2018)*. LNCS 10971. Springer, 2018, S. 155–163. URL: http://dbis.eprints.uni-ulm.de/1616/.

Name: Leon Ahmadi-Moghaddam Matrikelnummer: 1034441

**Erklärung**

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

Ulm, den **29.10.2020** L. Ahmadi

Leon Ahmadi-Moghaddam