FH JOANNEUM - University of Applied Sciences

BPMN2OWL – Speicherung von BPMN Modellen in Ontologien

Bachelorarbeit 2

zur Erlangung des akademischen Grades „Bachelor of Science in Engineering“

eingereicht am Studiengang Informationsmanagement

Verfasser:

Nikolaus Hribernig

Personenkennzahl: 1510422020

Betreuer:

FH-Prof. Mag. Dr. Robert Singer

Graz 2018

FH | JOANNEUM
University of Applied Sciences

Ehrenwörtliche Erklärung

Ich erkläre ehrenwörtlich, dass ich die vorliegende Bachelorarbeit selbstständig angefertigt und die mit ihr verbundenen Tätigkeiten selbst erbracht habe. Ich erkläre weiters, dass ich keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle ausgedruckten, ungedruckten oder dem Internet im Wortlaut oder im wesentlichen Inhalt übernommenen Formulierungen und Konzepte sind gemäß den Regeln für gutes wissenschaftliches Arbeiten zitiert und durch Fußnoten bzw. durch andere genaue Quellenangaben gekennzeichnet.

Die vorliegende Originalarbeit ist in dieser Form zur Erreichung eines akademischen Grades noch keiner anderen Hochschule vorgelegt worden. Diese Arbeit wurde in gedruckter und elektronischer Form abgegeben. Ich bestätige, dass der Inhalt der digitalen Version vollständig mit dem der gedruckten Version übereinstimmt.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

Nikolaus Hribernig Graz, Freitag, 18. Mai 2018

Kurzfassung

„Business Process Modelling Notation“ oder BPMN, als Standard um Prozesse zu modellieren, erfreut sich großer Beliebtheit. Viele verschiedene Hersteller bieten Editoren an, mit denen BPMN Prozesse modelliert werden können. Der Standard BPMN selbst ist in einem Dokument bestehend aus Beschreibungen in Fließtext und dazugehörigen Diagrammen der „Unified Modelling Language“ oder UML definiert. Unterschiedliche BPMN Editoren haben verschiedene Interpretationen des Standards BPMN. Diese Arbeit untersucht, wie BPMN Modelle in einer Ontologie des BPMN Standards gespeichert werden können. Die Speicherung soll im Rahmen der Modellierung stattfinden und ist der Grundschritt für eine Validierung des Modells mittels dieser Ontologie. Dadurch soll Standardkonformität eines BPMN Modells schon in der Entstehung gewährleistet sein. Fließtexte liefern Interpretationsspielraum und es finden sich Widersprüche mit den Diagrammen. Ontologie sind dabei ein Mittel um dieses Problem zu lösen. Diese Arbeit beschäftigt sich mit der Frage, ob BPMN Modelle während des Modellierens in einer Ontologie des Standards gespeichert werden können. Dazu werden verschiedene Umsetzungsmöglichkeiten untersucht, und eine Machbarkeitsanalyse im Rahmen einer „Feasibility Study“ durchgeführt. In dieser Arbeit kann aufgezeigt werden, dass eine Speicherung während des Modellierens nicht möglich ist. Dennoch ist die eine Speicherung, auch nach dem Modellieren ein Grundschritt für die Validierung des BPMN Modells mittels Ontologie. In der Umsetzung des Konzepts kann gezeigt werden, wo Probleme mit Standardkonformität auftreten und wie eine Ontologie diese lösen kann. Dadurch kann gezeigt werden, dass eine Formalisierung des Standards und der Modelle wichtig ist und eine Ontologie des Standards eine Möglichkeit ist dies durchzuführen.

Abstract

“Business Process Modelling Notation” or BPMN is a very popular standard for modelling processes. Many different manufacturers offer editors that can model BPMN processes. The standard itself is defined in a document consisting of descriptions in text and associated diagrams of “Unified Modelling Language” or UML. Different BPMN editors interpret the BPMN standard differently. This thesis, therefore, attempts to store BPMN models in an ontology of the BPMN standard during the modelling process. This storing is the basic step for validating the model with this ontology. As a result, the standard conformity of a BPMN model is ensured right after its creation. Texts leave room for interpretation and sometimes are contradicting to the UML diagrams. An Ontology, however, can resolve these contradictions. This thesis, therefore, deals with the question whether BPMN models can be stored in an ontology of this standard while modelling. For this purpose, various implementation options were examined, and a feasibility study was conducted. The results of this thesis show that storing the BPMN model while it is modelled is not possible. Nevertheless, the storage of the finished model is also a basic step in the validation of a BPMN model that uses an ontology.

The implementation of the concept presents difficulties in the standard conformance that can be solved using an ontology. As a conclusion, it can be shown that the formalization of the BPMN standard itself as well as its models is important and that the use of an ontology of this standard is a way to accomplish this formalization.

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abkürzung	Bedeutung
API	Application Programming Interface
BPM	Business Process Management
BPMN	Business Process Modelling Notation
IoT	Internet of Things
IT	Informationstechnologie
OMG	Object Modelling Group
OWL	Web Ontology Language
UML	Unified Modelling Language
XML	Extensible Markup Language

Abbildungsverzeichnis

1. Einleitung

Dieses Kapitel beschreibt die Motivation sowie die Problemstellung und Zielsetzung und Vorgehensweise dieser Arbeit. Außerdem wird die Forschungsfrage formuliert.

1.1. Motivation

„Business Process Modelling Notation“ oder BPMN, als Standard um Prozesse darzustellen, wurde 2011 in der zweiten Version veröffentlicht. Diese Version wurde in der bereichsübergreifenden Projektarbeit in einer Ontologie implementiert. Der Standard ist in Fließtext, sowie Diagrammen der „Unified Modelling Language“ oder UML beschrieben. Durch Widersprüche dieser zwei Arten und Interpretationsspielraum im Fließtext ergeben sich Probleme in der Einhaltung des Standards. Ontologien, wie in [Uschold, Gruninger 2004] beschrieben, widerspruchsfrei und ohne Interpretationsspielraum, eignen sich für diese Validierungen. Können konkrete BPMN Modelle in die im Rahmen der bereichsübergreifenden Projektarbeit entstandene Ontologie des BPMN Standards gespeichert werden, wäre dies ein Basisschritt für solch Validierung. Konformitätsverstöße in BPMN Modellen können vermieden werden, wenn bereits während des Modellierens validiert wird.

Die Forschungsfrage dieser Arbeit lautet daher, ob BPMN Modelle während des Modellierens in die Ontologie aus der bereichsübergreifenden Projektarbeit gespeichert werden können. Dieses Vorgehen kann als Basis zur Konformitätsprüfung zum Standard BPMN eines Modells mittels Ontologie während der Modellierung gesehen werden.

1.2. Problemstellung

Der Standard BPMN wird in einer aus Fließtext sowie UML Diagrammen bestehenden Dokumentation beschrieben. Fließtext sowie Widersprüche liefern Probleme bei der Formalisierung von BPMN. Des Weiteren bestehet der BPMN Standard aus Metadaten, die ebenfalls Einschränkungen vorweisen[1]. Viele Hersteller von BPMN Editoren interpretieren den Standard auf eigene Weise. Obwohl es Initiativen zur Angleichung der BPMN Modelle verschiedener Editoren gibt, garantieren diese nicht Widerspruchsfreiheit sowie Konformität gemäß dem Standard. Vor allem verpflichtend zu setzende Attribute einzelner Elemente können hierbei stark variieren[2]. In der bereichsübergreifenden Projektarbeit wurde eine Ontologie des Standards BPMN 2.0.2 erstellt, um den Standard weiter zu formalisieren und zu objektivieren. Nicht nur der Standard soll objektivierbar sein, sondern auch Modelle. Die Validierung von BPMN Modellen in der Ontologie aus der bereichsübergreifenden Projektarbeit bietet eine Möglichkeit, die Objektivierung der Modelle und die Konformität zum Standard zu

[1] Auf diese wird im Rahmen der weiteren Arbeit in Punkt 3.1 hingewiesen.

[2] Dies liegt an vererbten Attributen, wie in den Punkten 3.1. und 3.3 erläutert.

gewährleisten. Hierzu müssen Modelle zuerst in der Ontologie gespeichert und dann validiert werden.

1.3. Zielsetzung

Im Rahmen der bereichsübergreifenden Projektarbeit wurde die Ontologie des BPMN 2.0.2 Standards angefertigt. Zusätzlich wurde ein Konzept erstellt das eine Speicherung von konkreten BPMN Modellen in Ontologien zum Thema hat. Diese Speicherung dient dem Zweck, diese Modelle anschließend zu validieren. Diese Bachelorarbeit beschäftigt sich mit der Umsetzung dieses Konzepts auf Basis der erstellten Ontologie. Ziel ist es, auf einer vorhandenen Schnittstelle aufzubauen. Dies gilt sowohl für BPMN als auch für OWL als Standard für Ontologien. Daraus folgt, die Implementierung einer eigenen Schnittstelle ist nicht Teil dieser Arbeit. Die Forschungsfrage, ob die Speicherung von BPMN Modellen in Ontologien während des Modellierens möglich ist, muss dabei beantwortet werden. Können Prozesse schon während der Modellierung in der Ontologie gespeichert und im nächsten Schritt auf Standardkonformität geprüft werden, können inkompatible Modelle vermieden werden. Als Standard für diese Ontologien wird OWL verwendet. Des Weiteren werden Vorteile, die solch ein Vorgehen bietet, herausgearbeitet.

Die Arbeit soll die Frage beantworten, ob Speicherung von BPMN Modellen während des Modellierens, möglich ist.

1.4. Vorgehensweise

Zu Beginn folgt eine Reflexion der Ontologie und des Konzepts aus der bereichsübergreifenden Projektarbeit. Verschiedene Schnittstellen müssen untersucht werden. Daraus erflogt eine nachvollziehbare Entscheidung für eine bestimmte Schnittstelle. Darauf aufbauend wird ein Konzept entwickelt, um die in BPMN Diagrammen entstehenden Elemente mit Elementen aus der Ontologie des BPMN Standards in Zusammenhang zu bringen. Mithilfe dieses Konzepts wird der Prototyp einer Schnittstelle implementiert. Diese dient der Speicherung der Elemente aus den BPMN Modellen in der Ontologie. Es wird untersucht, ob die Speicherung in der Ontologie schon während des Modellierens erfolgen kann.

Zum Schluss dieser Arbeit wird das gewählte Vorgehen reflektiert, und die Forschungsfrage beantwortet. Diese lautet: Ist die Speicherung von BPMN Modellen in einer Ontologie während des Modellierens möglich ist?

1.5. Struktur der Arbeit

Zu Beginn der Arbeit erfolgen Definitionen. Anschließend werden Arbeiten beschrieben, auf denen diese basiert. Das Vorgehen in der bereichsübergreifenden Projektarbeit wird erklärt und das Konzept zur Implementierung einer Schnittstelle wird erläutert. Verschiedene Möglichkeiten, um BPMN oder Ontologien einzulesen und zu bearbeiten werden beschrieben, verglichen und Entscheidungen für jeweils eine getroffen. Schließlich wird das Konzept auf die Machbarkeit überprüft und umgesetzt. Am

Ende der Arbeit wird das Konzept getestet und Ergebnisse sowie Einschränkungen beschrieben.

2. Grundlagen

In diesem Kapitel werden Grundlagen erläutert sowie Begrifflichkeiten abgegrenzt. Die Standards BPMN und OWL werden beschrieben, sowie die Begriffe Geschäftsprozess und Ontologie definiert.

2.1. Geschäftsprozess

[Bucher, Winter 2009] beschreiben einen Prozess als definierte Abfolge von Aufgaben. Prozesse werden verwendet um aus definierten Inputs definierte Outputs zu erzeugen. Außerdem sind Prozessen Beschreibungen zugeordnet. Dies sind strukturierte Vorgaben zur Aufgabenausführung. Prozesse sind dabei auf Prozesskunden ausgerichtet, die unternehmensintern oder unternehmensextern sein können (vgl. [Bucher, Winter 2009]).

2.2. BPMN

Diese Arbeit beschreibt die Speicherung von BPMN Modellen in Ontologien. Dieser Punkt beschreibt den Standard und seine Elemente.

BPMN gilt als von der OMG definierter Standard, um Prozesse innerhalb eines Unternehmens zu verstehen, und diese grafisch darzustellen. BPMN Modelle werden im Standard XML gespeichert. Dennoch steht, wie in Abbildung 1 ersichtlich, eine grafische Repräsentation zur Verfügung. Mittels dieser ist es leichter möglich, Personen ohne BPMN Kenntnis den Ablauf eines Prozesses zu vermitteln (vgl. [BPMN 2014]).

BPMN Modelle bestehen aus verschiedenen Arten von Elementen. Dazu gehören unter anderem Events, Aktivitäten, Gateways und Flows[3].

- Events können dabei einen Prozess starten, beenden oder auch unterbrechen und wieder fortsetzen.
- Aktivitäten treten in verschiedenen Varianten auf. Wichtig hierbei sind Tasks und Unterprozesse[4]. Tasks sind aufführbar und können Eigenschaften wie erhalten einer Nachricht, senden einer Nachricht oder ausführen einer Aufgabe sein. Unterprozesse sind selbst Prozesse, die aber ein Teil eines anderen Prozesses sind. Unterprozesse können ausformuliert sein, oder als BlackBox dienen.
- Gateways kontrollieren den Fluss innerhalb eines Prozesses. Es gibt verschiedene Arten von Gateways, häufig sind exklusive- und parallele Gateways. Bei exklusiven Gateways wird eine Entscheidung getroffen, in der nur ein Pfad fortgeführt wird. Bei parallelen Gateway werden alle Flüsse gleichzeitig ausgeführt.

[3] Vgl. http://www.bpmnquickguide.com/ (16.05.2018)

[4] Im Standard BPMN werden Unterprozesse als „SubProcess“ bezeichnet.

- Flows können die Abfolge von Aktivitäten, Nachrichtenflüsse, oder Assoziationen darstellen (vgl. [BPMN 2014]).

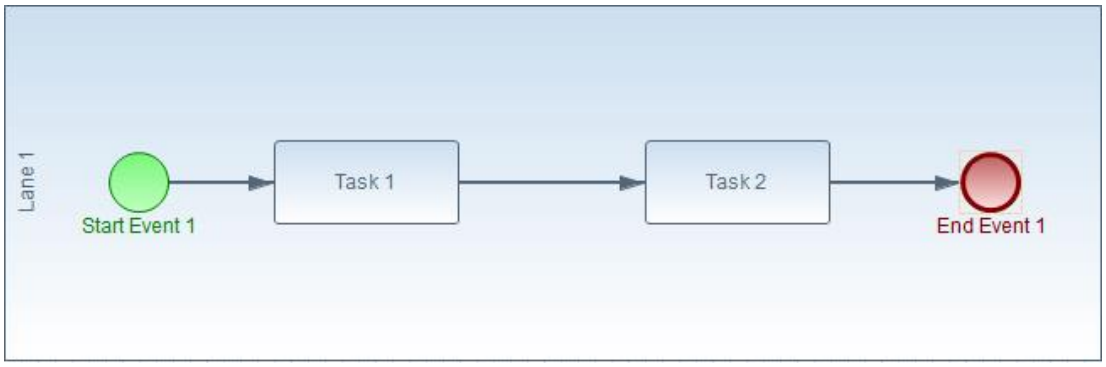

Abbildung 1: Ein Beispiel eines Prozesses in BPMN 2.0

Abbildung 1 zeigt ein Beispiel für einen mittels BPMN 2.0 modellierten Prozess. In diesem modellierten Prozess startet das Start Event den Prozess. Der Sequenzflow sieht als nächstes Task 1 vor. Nach Beendigung von Task 1 wird Task 2 ausgeführt. Danach endet der Prozess mit dem End Event.

2.3. **Feasibility Study**

In Kapitel 5 dieser Arbeit wird eine Machbarkeitsstudie des Konzepts durchgeführt. Dafür wird das Vorgehen einer „Feasibility Study“ gewählt.

„Feasibility Studies“ werden häufig bei neuen Ideen verwendet. Dabei wird die Fragestellung beantwortet, ob diese Idee rentabel ist und funktioniert. Eine „Feasibility Study“ bietet daher die Grundlage einer Entscheidung, ob ein Projekt umgesetzt wird. [Hofstrand, Holz-Clause 2009] spricht von fünf Punkten, die bei der Erstellung einer „Feasibility Study“ abgearbeitet werden sollten.

- Evaluieren von Alternativen: Zu Beginn werden mehre mögliche Wege untersucht und die Besten im Rahmen einer Vorstudie genauer analysiert.
- Vorstudie: Um nicht unnötig Ressourcen zu verwenden, können gute Möglichkeiten im Rahmen einer Vorstudie weiterbearbeitet werden. Sollte dieser Schritt positiv absolviert werden folgt die Marktstudie.
- Innovationsgrad: Im Rahmen einer Studie wird der Bedarf an einer Dienstleistung oder einem Produkt untersucht. Wird kein Bedarf festgestellt, muss die „Feasibility Study“ nicht fortgesetzt werden.
- Ergebnisse und Analyse: Bei der Analyse werden verschiedene mögliche Vorgehensweisen verglichen. Meist gibt es nicht eine perfekte Lösung, sondern Risiken und Möglichkeiten müssen abgewogen werden.

- Der fünfte Punkt ist die Entscheidung, die aufgrund der in der „Feasibility Study“ erarbeiteten Informationen getroffen wird (vgl. [Hofstrand, Holz-Clause 2009]).

2.4. **Ontologie**

Eine Ontologie wird in [Uschold, Gruninger 2004] als Spezifikation einer gemeinsamen Konzeptualisierung beschrieben, sowohl formal als auch explizit. Der Begriff Konzeptualisierung wird als Modell der Wirklichkeit definiert. Eine Ontologie kann aber nur einen speziellen Teil der Wirklichkeit abbilden. Innerhalb einer Ontologie werden explizite Namen und Definitionen vergeben. Hierbei beschreiben die Definitionen die Beziehungen unterschiedlicher Elemente. Durch die Formalität von Ontologien sind diese in einer Sprache ausgedrückt, die von Maschinen verstanden wird. Dadurch werden Interpretationsspielräume vermieden (vgl. [Uschold, Gruninger 2004]).

[Mädche et al 2001] beschreiben Anwendungsgebiete von Ontologien in der IT. So können Ontologien verwendet werden um gemeinsames Verständnis, beispielsweiße im „Supply Chain Management“, aufzubauen. Des Weiteren werden Ontologien im Wissensmanagement angewendet. Hierbei können nicht nur verschiedene Sichten auf Wissen zur Verfügung gestellt oder dynamisch generiert werden. Außerdem können mithilfe von Ontologien im E-Learning Bereich, durch Relationierung von Metadaten, Dokumente und Lehreinheit dynamisch zusammengestellt werden. Im Bereich des Semantic Webs, als Erweiterung des Internets durch semantische Modelle, können Ontologien verwendet werden um Daten untereinander in Beziehung zu setzen oder zu interpretieren (vgl. [Mädche et al 2001], S.2).

2.5. **Axiom**

Axiome spielen bei der Speicherung von Elementen in Ontologien eine große Rolle. Mittels Axiomen können in der in Punkt 6.2.2 beschriebenen OWL API Elemente in einer Ontologie gespeichert werden.

Axiome kommen aus der Mathematik, als ein Satz, der in einer Theorie nicht bewiesen werden muss. Er kann als wahr angesehen werden. Aus einem Axiom lassen sich Schlussfolgerungen ableiten, die dadurch auch als wahr angenommen werden können (vgl. [Hofmann 2015]).

Als Teil einer Ontologie beschreibt [Studer 2016] ein Axiom:

> *Axiome können Kardinalitäten, Wertebereiche oder auch Default-Werte von Relationen oder Attributen beschreiben, ebenso wie Eigenschaften von Beziehungen (vgl. [Studer 2016]).*

2.6. **OWL**

Die für diese Arbeit verwendete Ontologie, aus der bereichsübergreifenden Projektarbeit, ist im Standard OWL implementiert. Daher erfolgt unter diesem Punkt eine Beschreibung des Standards.

OWL, kurz für Web Ontology Language, ist der Ansatz der W3C Web Ontology Working Group um Web Content für Maschinen zugänglicher zu machen. Hierbei bietet der OWL 2 Standard nicht nur einige strukturelle Vorgaben, sondern auch eine definierte Syntax und eine definierte Semantik. Der aktuelle Standard, der für diese Arbeit verwendet wird, ist OWL 2. Dieser baut auf dem OWL 1 Standard auf, stellt allerdings auch Weiterentwicklungen zur Verfügung (vgl. [OWL 2012]).

2.7. **Protégé**

Protégé ist ein Ontologie Editor, der von der Stanford University entwickelt wurde. In diesem Editor können unter anderem Knoten, Beziehungen und Hierarchien abgebildet werden. Diese werden dann vom Editor in OWL umgewandelt. Zusätzlich können in Protégé auch Plugins inkludiert werden, die eine grafische Darstellung der Ontologie ermöglichen. Dadurch können Zusammenhänge von Elementen verstanden und abgebildet werden (vgl. [Stanford 2016]).

Protégé wurde sowohl in [Wageneder 2017] als auch in der bereichsübergreifenden Projektarbeit als Tool verwendet, die jeweils als Basis dieser Arbeit dienen.

2.8. **Klassen**

In Ontologien werden Elemente in Form von Klassen angelegt. Die Vererbung spielt dabei eine große Rolle.

Der Begriff der Klasse kommt in diesem Bezug aus der objektorientierten Programmierung. Eine Klasse beschreibt die Eigenschaften eines Objekts. Eine Klasse besitzt Attribute und Methoden, die diese Eigenschaften definieren. Dabei befindet sich eine Klasse in Hierarchien, die Attribute und Methoden vererben können (vgl. [Poetzsch-Heffter 2009], S.50ff, S.143f).

Als Beispiel hierfür zu nennen, wäre die Klasse Möbelstück mit den Attributen Höhe, Breite und Material. Subklassen von Möbelstück wären Sessel und Kasten, die jeweils die Attribute Höhe, Breite und Material der Klasse Möbelstück erben. Zusätzlich hat die Subklasse Sessel aber noch das Attribut Sitzfläche und die Subklasse Kasten noch das Attribut Einlagen. Abbildung 2 zeigt dieses Beispiel grafisch veranschaulicht.

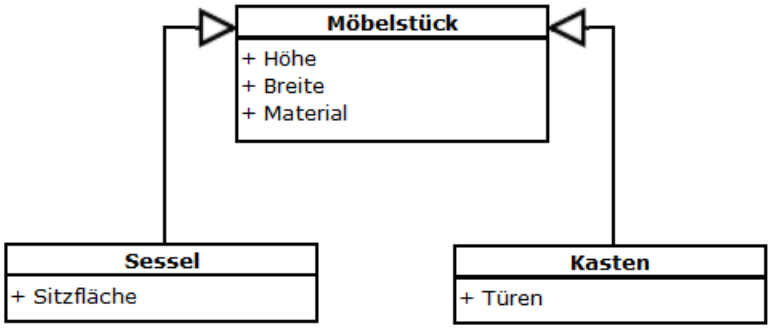

Abbildung 2: Die Klasse Möbelstück mit den Subklassen Sessel und Kasten

3. Aufbauende Arbeiten

Diese Arbeit beruht auf dem in der bereichsübergreifenden Projektarbeit erstellten Konzepts zur Speicherung von BPMN Modellen, in der im Rahmen dieser Projektarbeit erstellen Ontologie. In diesem Kapitel wird die Projektarbeit und die zugrundeliegende Methodik aus [Wageneder 2017] beschrieben.

3.1. Gründe für eine Ontologie des BPMN Standards

Der BPMN 2.0 Standard besteht aus einem 500 seitigen Dokument. In diesem Dokument wird mittels Fließtext und UML Diagrammen die Spezifikation beschrieben. Teilweise besteht zwischen dem Fließtext und den Diagrammen[5] auch ein Widerspruch (vgl. [BPMN 2014]).

Dadurch ist eine eindeutige Beantwortung zur Standardkonformität in vielen Fällen unmöglich. Auf dem Markt sind sehr viele unterschiedliche BPMN Editoren erhältlich. Da Widersprüche im Standard mehr Bewegungsfreiheit für Entwickler von Editoren bedeuten, erweitert das auch den Spielraum von den Modellen, die in den jeweiligen Editoren erstellt werden. Dies führt zu Inkompatibilität von Modellen der einzelnen Editoren[6].

Ein Grund für Standards ist, dass standardkonform Erstelltes immer gleich oder zumindest vergleichbar sein soll. Bestehen aber große Unterschiede zwischen Modellen eines gleichen Prozesses, erstellt in unterschiedlichen Editoren, ist dieser Vorteil eines Standards verloren. Dies wird ermöglicht durch Widersprüche im BPMN Standard oder nicht eindeutige Formulierungen im Fließtext. So können Anwender des Standards und Editoren Hersteller die jeweilige Interpretation wählen. Wie in Kapitel 4 gezeigt, sind Ontologien eine Möglichkeit, Wissen wie jenes aus dem BPMN Standard zu formalisieren und interpretationsfrei darzustellen. Als weiterer positiver Effekt kann die in Kapitel 2.4 beschriebene Maschinenverständlichkeit von Ontologien gesehen werden.

Es wurden bereits Versuche unternommen, den BPMN Standard in einer Ontologie abzubilden. [Rospocher et al 2014] beschreibt so einen Versuch. In diesem Ansatz wurde die gesamte Ontologie aus den XML Schemadateien des BPMN Standards generiert und manuell erweitert. Dies führt zum einen zu einer eigenen Struktur, außerdem ergeben sich dadurch gewisse Einschränkungen[7].

---

[5] Auf Widersprüche wird im Punkt 3.3 genauer eingegangen.

[6] Durch Initiativen wie der „Diagram Interchangability in BPMN2“ der OMG wird versucht, diese zu verbessern. Während die Interkompatibilität dadurch verbessert wird, gibt es noch immer Modelle von Editoren, die in andere Editoren nicht importiert oder exportiert werden können. Oder es treten Fehler in einem Editor auf, wie in Punkt 7.5.2 beschrieben.

[7] Wie in den Kapiteln 6 und 7 beschrieben wird, ist die Vererbung im BPMN Standard sehr heikel. In den Schemadateien des BPMN Standards ist diese sehr vereinfacht dargestellt. Dadurch fällt die Validierung für [Rospocher et al 2014] leichter, die Vererbung wird allerdings vernachlässigt.

Im Gegensatz dazu versucht eine Ontologie, die in [Wageneder 2017] erstellt und in der bereichsübergreifenden Projektarbeit reflektiert und erweitert wurde, den Standard zu formalisieren und eindeutig zu gestalten. Dadurch soll vor allem auch die Übersichtlichkeit verbessert werden, es dient also in erster Linie nicht dazu, um automatisierte Überprüfungen durchzuführen[8]. Nichts desto trotz, wird in dieser Arbeit die automatisierte Speicherung von BPMN Modellen implementiert, als Vorstufe der automatisierten Überprüfungen. Durch die unterschiedlich gesetzten Schwerpunkte ergeben sich Unterschiede in den Ontologien, vor allem die Struktur betreffend. Das in [Wageneder 2017] erstellte Konzept wird in Punkt 3.2 genau erläutert.

3.2. Konzept einer Ontologie des BPMN Standards

> *„Ziel dieser Arbeit ist, eine Art Nachschlagewerk zu schaffen, mit dessen Hilfe jedem Anwender die Modellierung in BPMN erleichtert werden soll. Sowohl der Einstieg als auch die kontinuierliche Hilfe beim Modellieren von Geschäftsprozessen stehen im Vordergrund.“ (vgl. [Wageneder 2017], S.9).*

[Wageneder 2017] beschreibt das Konzept einer Ontologie, in der der BPMN 2.0 Standard abgebildet ist. Als Tool der Wahl bei [Wageneder 2017] wurde Protégé gewählt. Ziel dieser Arbeit war, den Standard BPMN Version 2.0 in einer Ontologie darzustellen. In [Wageneder 2017] wurde das Konzept erstellt, die Ontologie wurde dann im Rahmen der bereichsübergreifenden Projektarbeit überarbeitet. Konzepte wurden hinterfragt und die Ontologie vervollständigt. Dieser Punkt beschreibt Basis Konzepte, die in der bereichsübergreifenden Projektarbeit übernommen wurden. Änderungen und Vervollständigung werden in Punkt 3.3 beschrieben.

Ein wichtiger Bestandteil bei [Wageneder 2017] ist, Elemente aus dem BPMN Standards mit Ontologie Elementen zu beschreiben. Als zentrales Konstrukt können hierbei Klassen verwendet werden. Die Hauptklasse, die in Ontologien verwendet wird, ist dabei „owl:Thing“. Alle hinzugefügten Klassen sind Subklassen von „owl:Thing“[9] und erben wie in Kapitel 2.8 beschrieben die Attribute dieser Hauptklasse. Elemente aus dem BPMN 2.0 Standard wurden in [Wageneder 2017] als Klassen dargestellt.

Verbindungen zwischen Klassen wurden als Objektverbindung dargestellt. In diesem Fall können diese Verbindungen als Attribute gesehen werden. In der in [Wageneder 2017] entstandenen und im Rahmen der bereichsübergreifenden Projektarbeit fertiggestellten Ontologie wurden diese nach dem Schema „has“ + Klassenname versehen oder, sofern im BPMN Standard explizit beschrieben, mit „has“ und dem jeweiligen Namen der Verbindung versehen. Die dritte Möglichkeit der Benennung einer Objektverbindung ist, dass zwei Verbindungen zwischen zwei Klassen bestehen. In

[8] Die Ontologie, die in der bereichsübergreifenden Projektarbeit erstellt wurde, soll der Validierung von Modellen dienen. Jedoch behandelt [Wageneder 2017] die Erstellung als Grundschritt. Diese Arbeit, mit der Speicherung von Modellen als weiterer Schritt, setzt die Vorarbeiten mit dem übergeordneten Ziel Validierung fort.

[9] Dies ist im OWL Standard definiert (vgl. [OWL 2012]).

diesem Fall wird die Objektverbindung mit „has“ und Zusatzinformationen und dem Klassennamen benannt.

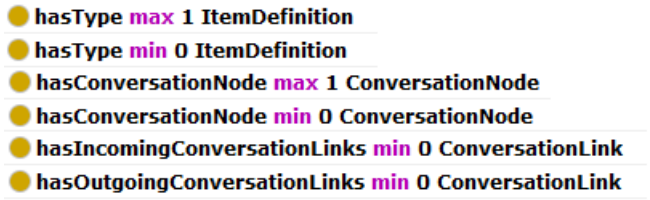

Abbildung 3: Mögliche Namenskonventionen von Objektverbindungen

Abbildung 3 zeigt alle drei möglichen Namenskonventionen von Objektverbindungen. „HasType“ als Verbindung zur Klasse „ItemDefinition“ wurde dabei im BPMN Standard so benannt. Ausdrücke wie „min“ und „max“ (in diesem Fall „min“ oder „max“, aber auch „exactly“, „some“ und „only“ sind in Protégé vorhanden[10]) und die dahinterliegende Zahl beschreiben die Kardinalität. Das Objekt benötigt also mindestens 0 aber maximal 1 „ItemDefinition“. „HasConversationNode“ zeigt die Standard Namenkonvention. Bei „hasIncomingConversationLinks“ und „hasOutgoingConversationLinks“ wurde eine Zusatzinformation verwendet, um einen eindeutigen Namen zu gewährleisten. Wie zuvor erwähnt ist in manchen Fällen im Standard die Verbindung zweier Klassen explizit benannt. [Wageneder 2017] zeigt wie Elemente des BPMN Standards in der Dokumentation des Standards beschrieben werden.

Elemente sind im Standard immer als Tabelle und als UML Diagramm vorhanden. Das Problem ist, dass diese teilweise nicht einheitlich sind. Auf dieses Problem wird in Punkt 3.3 genauer eingegangen. Verbindungen innerhalb der UML Diagramme sind ebenfalls manchmal benannt, manchmal nicht. Ist eine Verbindung explizit in Tabelle oder Diagramm benannt, wurde sie wie zuvor beschrieben auch so genannt. Abbildung 4 zeigt die Tabelle des BPMN Elements „ResourceParameter“ mit der expliziten Objektverbindung zum Element „ItemDefinition“, die in Abbildung 3 in der Ontologie zu sehen ist. Wie in der Tabelle, wurde auch in der Ontologie die Objektverbindung „hasType“ genannt und nicht etwa „hasItemDefinition“

Table 8.50 – ResourceParameter attributes and model associations

Attribute Name	Description/Usage
name: string	Specifies the name of the query `parameter`.
type: ItemDefinition	Specifies the type of the query `parameter`.
isRequired: boolean	Specifies, if a `parameter` is optional or mandatory.

Abbildung 4: Die Tabelle des BPMN Elements „ResourceParameter“ (entnommen aus [BPMN 2014], S.95)

[10] Im Rahmen der bereichsübergreifenden Projektarbeit und in [Wageneder 2017] wurden nur „min, „max“ und „exactly“ verwendet.

3.3. Vervollständigung der Ontologie des BPMN Standards im Rahmen der Bereichsübergreifenden Projektarbeit

Im Rahmen der bereichsübergreifenden Projektarbeit wurde die in [Wageneder 2017] begonnene Ontologie des BPMN Standards vervollständigt. Es wurde aber nicht wie in [Wageneder 2017] die Version 2.0 des BPMN Standards, sondern die Version 2.0.2 verwendet[11].

Im Rahmen dieser Arbeit konnte festgestellt werden, dass im BPMN Standard einige widersprüche herrschen, so musste ein Vorgehen entwickelt werden, um den Standard Widerspruchfrei abzubilden. Da [Wageneder 2017] ein Konzept entwickelte, nicht aber alle Elemente implementierte, wurden diese Widersprüche nicht berücksichtigt. Die in 3.2 vorgestellte Namensgebung wurde beibehalten. Die Struktur der Ontologie wurde geändert, um allen in BPMN inkludierten Elementen zu entsprechen. Des Weiteren wurden auch viele zweiseitige Beziehungen hinzugefügt. Da die Struktur größtenteils verändert wurde, veränderten sich auch die vererbten Attribute der Elemente.

Vor allem zwischen UML Diagrammen und den in Abbildung 4 gezeigten Tabellen herrschen widersprüche. Abbildung 5 ist ein im BPMN Standard verwendetes UML Diagramm. Dieses zeigt deutlich im markierten Bereich, dass die Verbindung „type“ zwischen „CorrelationProperty“ und „ItemDefinition“ existiert. Des Weiteren zeigt Abbildung 5 das Problem von UML Diagrammen, sehr viel Information auf nur begrenztem Raum darzustellen. Mehrere Verbindungen sind nur schwer einzelnen BPMN Elementen zuordenbar. Gewisse Verbindungen sind aufgrund des Platzmangels sogar übereinandergeschrieben und so nur sehr schwer bis gar nicht lesbar. Es ist in Abbildung 5 deutlich zu erkennen, dass zwischen „CorrelationProperty“ und „ItemDefinition“ die Verbindung „type“ ist. Obwohl in der Ontologie der Name der Objektverbindung immer mit „has“ und dem Namen der Klasse mit, der die Verbindung besteht gewählt wurde, ist dies wie in 3.2 beschrieben nicht der Fall. Dennoch müsste es in der Tabelle, die die Attribute der Klasse „CorrelationProperty“ zeigt, ein Attribut mit dem Namen „type“ geben. Als Attributtyp muss dieses Attribut laut UML Diagramm und UML Standard die Klasse „ItemDefinition“ haben.

[11] Der Standard BPMN 2.0 wurde im Jahr 2011 herausgegeben, 2.0.2 erst im Jahr 2013.

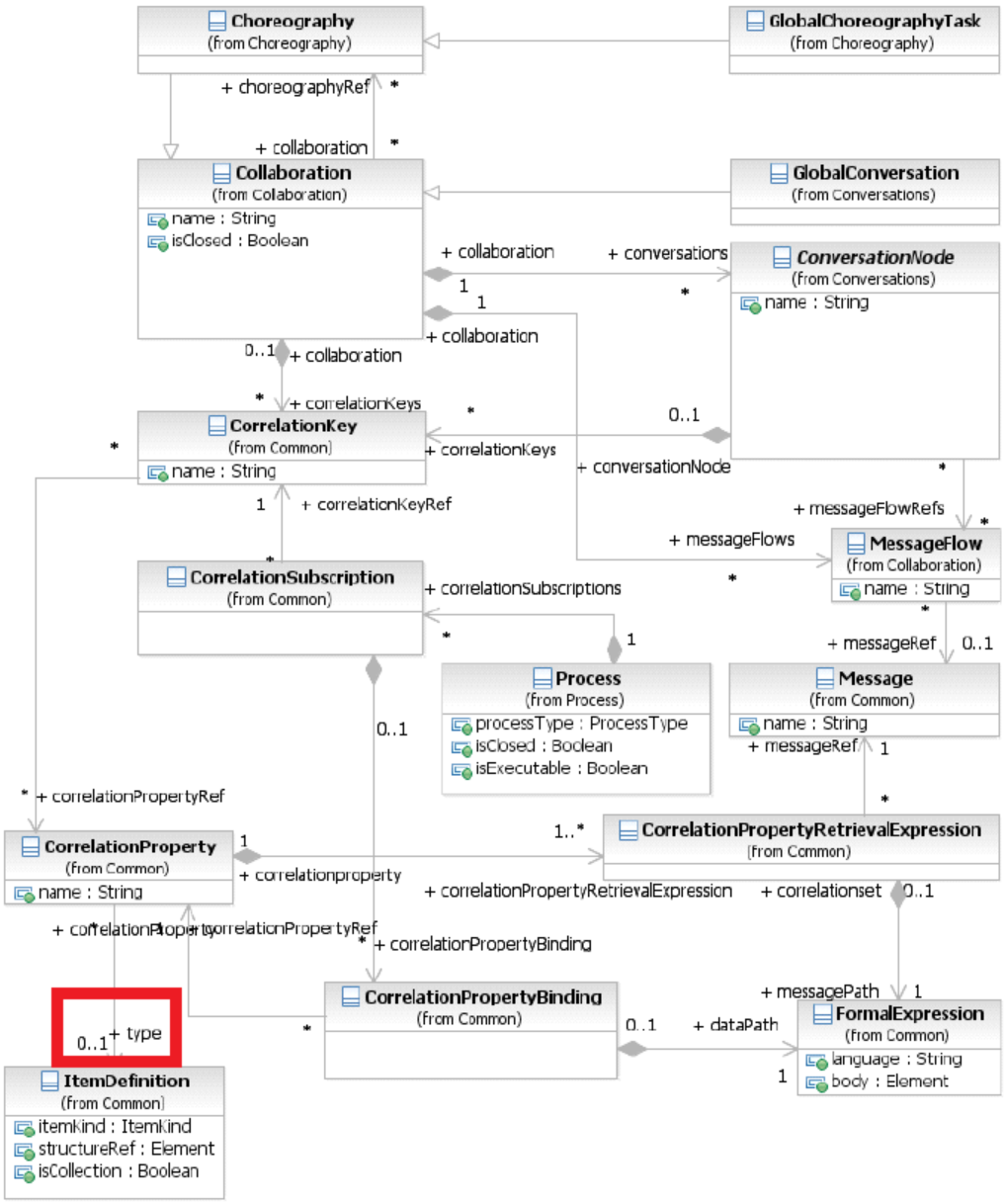

Abbildung 5: Ein UML Diagramm aus dem BPMN Standard (entnommen aus [BPMN 2014], S.74)

Dem widerspricht Abbildung 6, die zugehörige Tabelle zu dem in Abbildung 5 gezeigten UML Diagramm. Wie rot eingerahmt in Abbildung 6 zu erkennen ist, ist „type“ vom Typ „string“. Laut dem Diagramm müsste diese Verbindung aber ein Attribut des Typs „ItemDefinition“ sein. Vergleichbar ist das Attribut „type“ mit dem Attribut „correlationPropertyRetrievalExpression“. Dieses Attribut hat wie im UML Diagram beschrieben den korrekten Typ der Klasse „CorrelationPropertyRetrivalExpression“. Wäre „type“ ein einfacher String, müsste es wie „name“ im UML Diagramm in Abbildung 5 eingezeichnet sein. Dies zeigt einen eindeutigen Widerspruch zwischen dem UML Diagramm in Abbildung 5 und der Tabelle in Abbildung 6. Dadurch wird deutlich, dass eine widerspruchsfreie und eindeutige Definition des gesamten BPMN Standards nötig ist. Dadurch wird die Wichtigkeit der in [Wageneder 2017] und der be-

reichsübergreifenden Projektarbeit erstellten Ontologie des BPMN Standards gezeigt. Speicherung von konkreten BPMN Modellen in Ontologien ist ein weiterer Schritt in Richtung objektiviertem und eindeutigem Standard. Diese Arbeit beruht auf dem Konzept zur Speicherung von BPMN Modellen in Ontologien, das ebenfalls in der bereichsübergreifenden Arbeit erstellt wurde und in Punkt 3.4 genauer erläutert wird.

Table 8.32 – CorrelationProperty model associations

Attribute Name	Description/Usage
name: string [0..1]	Specifies the name of the `CorrelationProperty`.
type: string [0..1]	Specifies the type of the `CorrelationProperty`.
correlationPropertyRetrieval-Expression: CorrelationPropertyRetrievalExpression [1..*]	The `CorrelationPropertyRetrievalExpressions` for this `CorrelationProperty`, representing the associations of `FormalExpressions` (extraction paths) to specific `Messages` occurring in this **Conversation**.

Abbildung 6: Die Tabelle zum UML Diagramm aus Abbildung 5 (entnommen aus [BPMN 2014], S.75)

3.4. Konzept zur Speicherung von BPMN Modellen in der Ontologie des BPMN Standards

Wie bereits in den Punkten 2.9 und 3.3 erwähnt, war ein Teil der bereichsübergreifenden Projektarbeit die Erstellung eines Konzeptes zur Speicherung von BPMN Modellen in der erarbeiteten Ontologie. Das Team beschränkte sich dabei rein auf das Konzept, die Umsetzung ist Teil dieser Arbeit und wird im Kapitel 6 genauer beschrieben. Die Überlegungen des Konzepts waren in diesem Fall nur theoretischer Natur. Dennoch wurde, um das in Abbildung 7 links ersichtliche BPMN Diagramm zu erzeugen, das BPMN2 Plugin für Eclipse[12] verwendet. Dieses Diagramm ist allerdings eine grafische Repräsentation, um das Konzept zu beschreiben. Die Implementation war nicht Teil der bereichsübergreifenden Projektarbeit. Rechts ist die im Rahmen der bereichsübergreifenden Projektarbeit entstandene Ontologie des BPMN 2.0.2 Standards zu sehen, im Tool Protégé. Um die beiden Bestandteile zu verbinden, müssen zwei Aufgaben erfüllt werden: Zum Ersten, die Sammlung der Informationen aus dem BPMN Diagramm. Hierbei werden alle Informationen über die verwendeten Elemente sowie die konkreten Attribute gesammelt. Im Bild ist dies durch „BPMN2OWL“ dargestellt. Als zweites Element wird die „OWL API“[13] gezeigt.

[12] Da die Implementation nicht Teil der bereichsübergreifenden Projektabeit war, war die Überlegung, mit welchem Tool Zugriff auf sowohl ein BPMN Diagramm als auch auf eine Ontologie möglich ist. BPMN Editoren bieten zwar Veränderungen des Editors im Sinne von „CustomTasks“ an, allerdings benötigt es eine Entwicklungsumgebung oder zumindest Integrationen in eine solche, um beides zu adressieren. Eclipse schien daher als beste Möglichkeit für die Umsetzung und BPMN2-Modeler mit sowohl der besten Dokumentation als auch den kürzesten Veröffentlichugsintervallen.

[13] Der Name „OWL API“ ist hierbei abstrakt zu sehen. Konkrete Instanzen einer „OWL API“ werden in Punkt 6.2 erläutert.

Für Protégé gibt es eine eigene Java API, für die auch ein Wiki erzeugt wurde. Diese Protégé API kann als konkrete Instanz der in Abbildung 7 gezeigten „OWL API“ gesehen werden. Dadurch ist es möglich, auf Ontologien zuzugreifen und diese zu verändern. Auf dieser Möglichkeit basiert das in der bereichsübergreifenden Projektarbeit erstellte Konzept. Dies garantiert, dass Informationen, aus einem in einem Editor erstellten Diagramm, zu den jeweiligen Elementen in der Ontologie zugeordnet werden können.

Abbildung 7 zeigt zusätzlich, dass Informationen aus der Ontologie verwendet werden können, um das BPMN Diagramm zu validieren. Diagrammelemente können auf alle notwendigen Attribute und Verbindungen untersucht werden. Die Ontologie ermöglicht dabei, dass die Validierung nicht mehr Editoren abhängig ist. Abbildung 7 geht dabei über den Rahmen dieser Arbeit hinaus. Es zeigt nicht nur die Speicherung von BPMN Modellen in Ontologien, sondern auch wie Ontologien zur Validierung genützt werden könnten. Während die Validierung von BPMN Elementen nicht mehr Teil dieser Arbeit ist, kann die Validierung während des Modellierens als übergeordnetes Ziel gesehen werden.

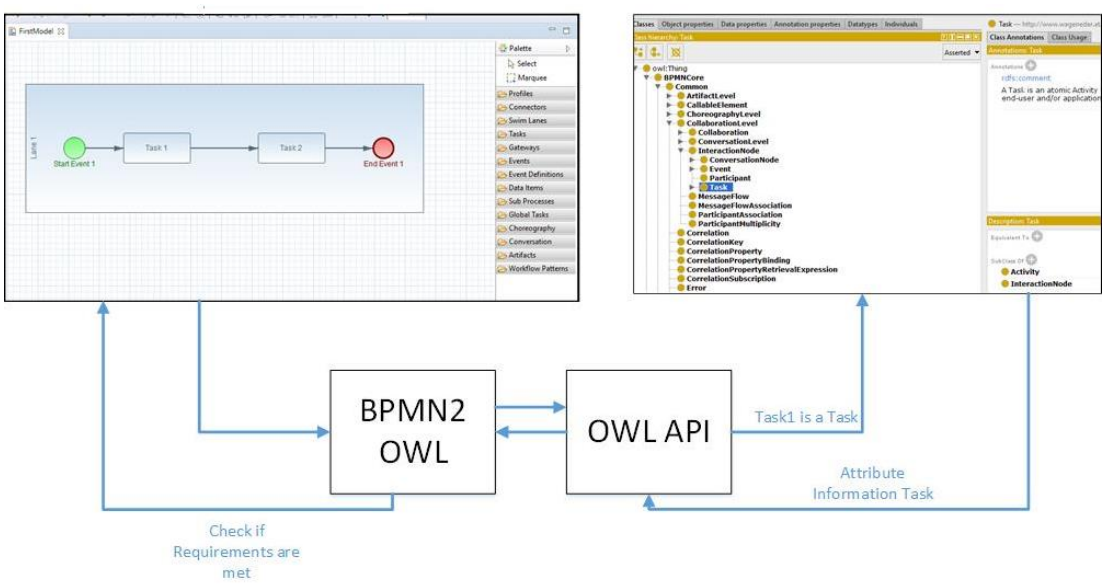

Abbildung 7: Das in der bereichsübergreifenden Projektarbeit erstellte Konzept

4. BPMN Editoren

Dieses Kapitel zählt verschiedene BPMN Editoren auf. Dabei wird darauf geachtet, dass der jeweilige Editor auch eine programmierbare Schnittstelle aufweist. Gegen Ende des Kapitels werden Vor – und Nachteile kritisch reflektiert und eine Entscheidung getroffen. In dieser Arbeit soll nicht nur ein Prozessmodell erstellt, eingelesen, oder verändert werden, sondern auch eine Ontologie. Daher ist ein entscheidendes Kriterium bei der Toolauswahl, dass das Tool in eine Entwicklungsumgebung eingebunden ist, oder eine API oder ein Plugin enthalten die das zu ermöglichen scheint.

4.1. Eclipse mit BPMN2-Modeler Plugin

Eclipse ist eine Open Source Software, die als Open Development Platform für verschiedenste Software dient. Geleitet wird das Eclipse Projekt von der Eclipse Foundation. Die Eclipse Foundation ist eine Non-Profit Organisation, die sowohl private Mitglieder, als auch Organisationen als Mitglieder hat. Das Eclipse Projekt startete 2001[14].

Um grafische BPMN Modelle erstellen zu können wird das BPMN2-Modeler Plugin benötigt.

Mithilfe des BPMN2-Modeler Plugins können grafische Modelle eines Prozesses erstellt werden, die dem BPMN 2.0 Standard entsprechen. BPMN Modelle werden im XML Format gespeichert. Da die Anzahl an Anforderungen an die Modellierung von Prozessen immer weiter steigt, ist eine grafische Darstellung nötig, um die Verständlichkeit zu erhöhen. Das BPMN2-Modeler Plugin ermöglicht die Erstellung grafischer Modelle innerhalb der Eclipse Software. Neben der Möglichkeit Verhalten und User Interface des Editors zu verändern, kann mittels Java Code auch auf Elemente der Modelle zugegriffen werden[15].

4.2. Camunda Modeler

Ebenso wie Eclipse ist auch Camunda eine Open Source Plattform. Im Gegensatz zu Eclipse ist es aber auf Business Process Management, kurz BPM, ausgelegt. Deshalb bietet Camunda standardmäßig alle für BPMN 2.0 nötigen Funktionen. Es muss kein extra Plugin heruntergeladen werden. Des Weiteren stellt Camunda eine BPMN API zur Verfügung, die verwendet werden kann, um Elemente auszulesen und zu manipulieren. Dabei werden die XML Dateien ausgelesen. Camunda bietet auch ein Eclipse Plugin an, jedoch wird dies laut Camunda Docs nicht mehr empfohlen.[16]

---

[14] Vgl. http://www.eclipse.org/org/ (16.05.2018)

[15] Vgl. https://www.eclipse.org/bpmn2-modeler/ (16.05.2018)

[16] Vgl. https://docs.camunda.org/manual/7.8/ (16.05.2018)

4.3. NetBeans jBPMN Plugin

Auch für den Editor NetBeans ist ein BPMN Plugin verfügbar. NetBeans ist die, laut NetBeans, offiziell empfohlene IDE für Java 8. Ebenso wie Eclipse ist auch NetBeans für mehrere Plattformen und Programmiersprachen geeignet. Deshalb wird ein zusätzliches Plugin namens jBPMN benötigt[17].

Mithilfe des jBPMN Plugins können innerhalb des NetBeans Editors Modelle, die dem BPMN Standard entsprechen, erstellt werden. Allerdings können Modelle nur grafisch verändert werden (vgl. [Gupta 2017]).

4.4. yEd

yEd ist ein Tool, um grafische Diagramme möglichst gut darstellen zu können. Eine mögliche Diagrammart ist hierbei BPMN. Bei yEd können XML Dateien, die Standards wie BPMN entsprechen, importiert werden. Zusätlich bietet yEd hilfreiche Features wie ein automatisches Arrangieren von Diagrammelementen an[18]. Auch yEd bietet eine API an. Mit dieser API können Diagramme erstellt und „gezeichnet“ werden[19]. Es gibt keine Möglichkeit bestehende Modelle und Diagramme zu erstellen oder zu verwalten[20].

[17] Vgl. https://netbeans.org/features/index.html (14.12.2017)

[18] Vgl. https://www.yworks.com/products/yed (16.05.2018)

[19] Vgl. https://docs.yworks.com/yfilesjavafx/doc/api/#/home (16.05.2018)

[20] Dies unterscheidet yEd von BPMN2 Modeler Plugin und Camunda.

5. Feasibility Study

In diesem Kapitel wird die durchgeführte „Feasibilty Study“ beschrieben. Diese wird wie im Punkt 2.3 definiert durchgeführt. Die Studie dient der Machbarkeit. Am Ende dieses Kapitels soll die Entscheidung stehen, ob BPMN Prozesse während des Modellierens in eine Ontologie gespeichert werden können.

5.1. Evaluieren von Alternativen

Um dieses Ziel zu erreichen, können mehrere Tools verwendet werden. Diese Tools für die Umsetzung sind in Kapitel 4 beschrieben.

5.2. Vorstudie

Weil Eclipse mit BPMN2-Modeler die Möglichkeit eines „Plugin im Plugins“ bietet, eignet es sich am besten für die Umsetzung der Zielvorgabe. So können eigene Elemente entwickelt, Validierungsvorschriften erstellt und mittels java Code in Eclipae direkt bearbeitet werden. Camunda hat hierbei zwei Einschränkungen. Zwar bietet der Camunda Modeler auch die Möglichkeit von Validierungsregeln, die zur direkten Speicherug verwendet werden könnten. Allerdings kann mittels java Code nicht darauf zugegriffen werden. Wird die API von Camunda verwendet, kann ein Modell eingelesen werden, aber nicht während des Modellierens. Dennoch bietet die API von Camunda hilfreiche Funktionen für die Implementation[21].

Die Einschränkungen von yEd und NetBeans wurden bereits in den Punkten 4.3 und 4.4 erläutert.

Da zusätzlich die Dokumentation für BPMN2-Modeler sehr gut ist, wird die Umsetzbarkeit mit diesem Tool geprüft. Ob die Umsetzung wie beschrieben möglich ist, wird im Rahmen des Punktes 5.3 überprüft, sowie im Punkt 5.4 analysiert.

5.3. Innovationsgrad

[Di Martino et al 2016] bietet bereits eine Grundlage für einen BPMN2OWL Converter. Allerdings wurde damals als Grundlage „The BPMN 2.0 Ontology“ von [Rospocher et al 2014] verwendet. Im Rahmen dieser Arbeit wird auf die in der Bereichsübergreifenden Projektarbeit entstandenen Ontologie zurückgegriffen. Auf die Unterschiede dieser beiden Ontologien wurde in Punkt 3.1 eingegangen.

Das Ziel dieser Arbeit ist, BPMN Modelle während des Modellierens in einer Ontologie zu speichern und so eine Validierung vorzubereiten. Während es in Editoren üblich ist, dass Modelle während der Modellierung validiert werden, stellt sich die Frage inwiefern diese Validierung beeinflusst werden kann. Daher muss überprüft werden, ob dieser Ansatz mit den vorhandenen Editoren durchführbar ist. In Kapitel 4 wurden

[21] Beispielsweise können Elementattribute durch Hilfsfunktionen direkt adressiert werden.

bereits mehrere Editoren mit Vorteilen, Nachteilen sowie Einschränkungen erläutert. Dadurch hat sich herausgestellt, dass zur Umsetzung dieses Ansatzes BPMN2-Modeler Plugin für Eclipse am besten geeignet ist.

Das Plugin für Eclipse ist simpel über den Eclipse Marketplace downloadbar. Als Version in dieser Arbeit wird „Eclipse IDE for Java Developers“ Version Oxygen.3 Release (4.7.3) verwendet. Zugehörig ist BPMN2 Modeler in der Version 1.4.2. Wie in Abbildung 8 ersichtlich können Zusatzelemente wie eine Beispielsammlung dazu geladen werden.

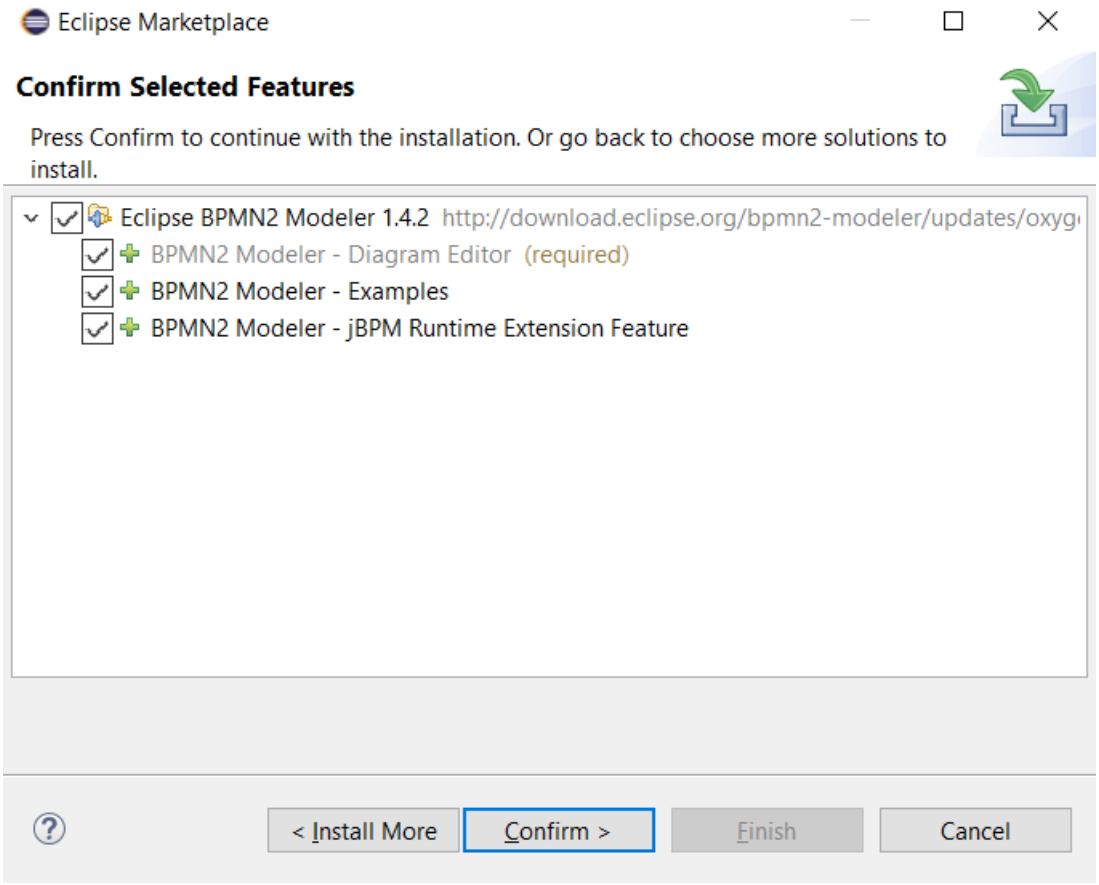

Abbildung 8: Möglichkeiten bei der Installation des BPMN2 Modeler Plugins

Die Beispiele sind sehr hilfreich, um die Funktionen des Plugins zu verstehen und sich einen Eindruck über die Anwendung zu verschaffen. Das jBPM[22] Plugin bietet eine Möglichkeit, die erstellten Prozesse des Modelers ausführbar zu machen.

[22] jBPM ist eine Suite an Programmen die sich mit Business Process Management beschäftigt. Ein Hauptziel dieser Anwendungen ist die Verbindung der technischen Analystensicht und der Prozessmanagement Sicht. jBPM bietet eine Workbench, kann aber, wie in diesem Fall, auch in Eclipse verwendet werden. Dadurch wird ermöglicht modellierte Prozesse ausführbar zu machen (vgl [Red Hat]). Außerdem bietet jBPM in Eclipse Möglichkeiten um Prozessmodelle einzulesen.

Nach der Installation des BPMN2 Plugins können leere BPMN Modelle, generische BPMN Diagramme, mit Inhalten zur Orientierung oder ausführbare jBPM Prozessdiagramme[23] ausgewählt werden.

In allen 3 Fällen werden Dateien erstellt, die entweder im Diagram Editor grafisch angezeigt und bearbeitet oder in einem Texteditor als XML Datei bearbeitet werden können. Alle dadurch erzeugten Prozesse werden durch eine im Plugin integrierte Validierung überprüft. Allerdings ist es auf diese Weise nicht möglich, eigene Elemente für die Prozessmodellierung[24] anzulegen. Diese sind durch das Plugin definiert.

Das BPMN2 Modeler Plugin für Eclipse bietet an, das Plugin mit selbst entwickelten Plugins zu Erweitern. So ist man nicht auf die vom BPMN2-Modeler Plugin bereitgestellte Palette an Elementen beschränkt.

Um ein Plugin zur Erweiterung von BPMN2-Modeler Plugin für Eclipse zu schreiben, wird eine umgehende Hilfestellung im BPMN2-Modeler Wiki[25] angeboten. Dies reicht von Änderungen an der Runtime, die benötigt wird um das selbst implementierte Plugin auszuführen, über Änderung und Adaptierung von Elementen zur Erstellung neuer Elemente und der Implementierung von Validierungen dieser im BPMN2 Modeler.

Im Rahmen dieser Arbeit wurden viele Anleitungen des BPMN2-Modeler Wiki bearbeitet. So konnte ein Plugin erstellt werden, dass auf neue Elemente zugreifen, sowie in einer eigens bearbeiteten Runtime mit eigenem Namespace bearbeitet werden konnte. Des Weiteren konnten diese Elemente mit Default Werten befüllt und so User- und Userinnenfreundlicher gestaltet werden. Obwohl dies im Praxisfalle sehr hilfreich und praktisch für die Adaptierung auf bestimmte Bedürfnisse ist, kann nicht, wie für die Aufgabenstellung dieser Arbeit benötigt, auf die Informationen der Elemente zugegriffen werden. Diese können zwar default gesetzt, aber dann nicht mehr ausgelesen werden.

Wie bereits in Abbildung 8 gezeigt ist es möglich, das BPMN2-Modeler Plugin mit jBPM zu verbinden. Mit jBPM können dabei Modelle mit dem Diagram Editor erstellt und dann in einer Runtime Environment geladen werden[26]. Dadurch wird ermöglicht, auf einzelne Diagrammteile zu verweisen. Ebenso können Informationen aus dem Diagramm so abgefragt und bearbeitet werden. In Grundzügen ist dies die Funktionalität, die für diese Arbeit benötigt wird.

Der Ansatz war, beim Erstellen eines Elementes in der grafischen Oberfläche eine Referenz zu jenem Element zu erhalten, welches dann mittels java Code in die OWL

[23] Ausführbare jBPM Prozessdiagramme unterscheiden sich von generischen BPMN Diagrammen durch das Schema der XML Diagramme. In diesem Fall sind sie auf die Verwendung von jBPM optimiert. Während aber das ausführbare jBPM Prozessdiagramm wie das leere BPMN Modell im Format „.bpmn2“ gespeichert werden, werden generische BPMN Diagramme als „.bpmn“ File gespeichert.

[24] Wie beispielsweise ein selbst erstellter „Task“ Typ, mit eigens bestimmten Attributen.

[25] Vgl. https://wiki.eclipse.org/BPMN2-Modeler/DeveloperTutorials (16.05.2018)

[26] Vgl. https://docs.jboss.org/jbpm/v6.0/userguide/jBPMCoreEngine.html (16.05.2018)

gespeichert werden kann. Dieser Ansatz hat sich als falsch und unmöglich herausgestellt. Alle Informationen über ein BPMN Modell werden im „.bpmn“ file im XML Format gespeichert. Um auf die Informationen aus einem Modell zuzugreifen, muss daher entweder direkt oder indirekt dieses XML herangezogen werden.

Eine bessere Möglichkeit wäre über die Klasse „Validation“, die im BPMN2-Modeler Wiki beschrieben ist[27]. Dieses verwendet als Superklasse den Eintrag „AbstractModelConstraint“. Da dieser Eintrag sehr alt ist, konnte bereits eine Verbesserung dieses Interfaces gefunden werden, die Klasse „IBPMN2ElementValidator“. Allerdings sind Implementationen dieser Klasse sehr aufwendig und würden daher den Rahmen dieser Arbeit sprengen[28].

```
48                String taskName = null;
49                if (object.getIoSpecification()!=null) {
50                        for (DataInput di : object.getIoSpecification().getDataInputs()) {
51                                if ("TaskName".equalsIgnoreCase(di.getName())) { //$NON-NLS-1$
52                                        for (DataInputAssociation dia : object.getDataInputAssociations()) {
53                                                if (dia.getTargetRef() == di) {
54                                                        if (dia.getAssignment().size()!=0) {
55                                                                Assignment a = dia.getAssignment().get(0);
56                                                                if (a.getFrom() instanceof FormalExpression) {
57                                                                        String body = ((FormalExpression)a.getFrom()).getBody();
58                                                                        if (body!=null && !body.isEmpty()) {
59                                                                                taskName = body;
60                                                                                break;
61                                                                        }
62                                                                }
63                                                        }
64                                                }
65                                        }
66                                }
67                        }
68                }
```

Abbildung 9: Funktion um Namen eines „UserTask" zu adressieren[29]

Um den Namen eines „UserTask“ mittels Camunda API zu adressieren, wird der Befehl „getName()“ verwendet. Die gezeigte Implementation kann als gut und als Referenz gesehen werden, da sie von der offiziellen „github“ Seite des BPMN2-Modeler Plugins für Eclipse kommt.

5.4. Ergebnisse und Analyse

Wie in Kapitel 4 beschrieben, gibt es vier für diese Arbeit relevante Anbieter von BPMN Editoren. Obwohl es unzählige Anbieter von BPMN Editoren gibt, sind es diese, die zusätzlich Interfaces anbieten. Mit der Hilfe dieser Interfaces soll es möglich auf die in BPMN Modelle zuzugreifen und gleichzeitig in der Ontologie zu speichern. Da

[27] Vgl. https://wiki.eclipse.org/BPMN2-Modeler/DeveloperTutorials/Validation (16.05.2018)

[28] Vgl. https://github.com/eclipse/bpmn2-modeler/blob/master/plugins/org.eclipse.bpmn2.modeler.core/src/org/eclipse/bpmn2/modeler/core/validation/validators/AbstractBpmn2ElementValidator.java (16.05.2018)

[29] Vgl. https://github.com/eclipse/bpmn2-modeler/blob/e8a07cacf2db97e5a66ffc47dbfbb8865717905b/plugins/org.eclipse.bpmn2.modeler.runtime.jboss.jbpm/src/org/eclipse/bpmn2/modeler/runtime/jboss/jbpm5/validation/validators/UserTaskValidator.java (16.05.2018)

das Ziel dieser Arbeit ist, während der Modellierung BPMN Elemente in der Ontologie aus der bereichsübergreifenden Projektarbeit zu speichern, konnten drei der vier Toolmöglichkeiten bereits in Punkt Vorstudie ausgeschlossen werden. Obwohl es im BPMN2-Modeler Plugin für Eclipse möglich ist, ein Plugin zur Erweiterung des BPMN2-Modeler Plugins zu implementieren, kann im Rahmen dieser Plugins nicht auf konkrete Elementdaten zugegriffen werden. Wie im BPMN2-Modeler Wiki beschrieben, gibt es viele Möglichkeiten um den Modeler auf eigene Bedürfnisse anzupassen. So gibt es auch die Variante einen eigenen Validator zu implementieren. Dieses ist allerdings, wie in Punkt 5.3 beschrieben, sehr umständlich und würde den Rahmen dieser Arbeit sprengen.

Das bedeutet, dass BPMN Modelle im Rahmen dieser Arbeit zwar in einer Ontologie gespeichert werden können, allerdings nicht im Laufe der Modellierung. Da wie in Punkt 5.2 beschrieben Camunda nützliche und hilfreiche Funktionen bietet, wird diese API verwendet. Auf die Verwendung des Camunda Modelers wird verzichtet, da wie in Kapitel 4 erwähnt wurde, die Elemente mittels java in die Ontologie gespeichert werden müssen.

6. Konzept zur Speicherung von BPMN Modellen in Ontologien mittels XML Dateien

Das Ziel dieser Arbeit ist es, BPMN Modelle in Ontologien zu speichern. Das Hauptziel war, dies während der Modellierung zu ermöglichen. Wie bereits in Kapitel 5 mittels einer „Feasibility Study“ untersucht, ist dieses Ziel im Rahmen dieser Arbeit nicht möglich. Daher wird auf fertige BPMN Prozesse und deren XML Dateien mittels der Camunda API[30] zugegriffen. In diesem Kapitel wird eine Methodik erstellt, wie BPMN Modelle mittels XML Dateien eingelesen und dann mit den Eigenschaften in die in der bereichsübergreifenden Projektarbeit erstellte Ontologie gespeichert. Um die Funktion der Methodik zu beweisen, werden BPMN Modelle mit Level 1 Elementen angewendet und in der Ontologie gespeichert.

6.1. Einlesen von BPMN Modellen

Um BPMN Modelle mithilfe der Camunda API zu importieren muss diese erst in konkrete Projekte importiert werden. Dies wird durch Maven[31] ermöglicht. Die Camunda API stellt die Klasse „BpmnModelInstance“ zur Verfügung um BPMN Modelle einlesen zu können. Die Bibliothek „org.camunda.bpm.model.bpmn.Bpmn“ bietet die Methode „readModelFromFile()“. Dadurch stehen die Informationen aus dem BPMN Modell zur Verfügung. Um, wie in dieser Arbeit vorgesehen, alle Elemente eines bestimmten Typs aus dem BPMN Modell auslesen, werden zusätzlich noch ein Model aus der Bibliothek „org.camunda.bpm.xml“ und die Klasse „ModelElementInstance“ aus derselben Bibliothek benötigt. Das Model kann dann durch die „BPMNModelInstance“ adressiert werden und über dieses Model kann eine Collection[32] des angegebenen Typs abgefragt werden.

Diese Methodik wurde in dieser Arbeit für alle verlangten Elemente berücksichtigt.

6.2. Speicherung in Ontologien

Neben dem Erfassen von Daten aus vorhandenen BPMN Modellen, das in den Kapiteln 4 und 5 schon ausführlich beschrieben wurde, liegt ein Hauptaugenmerk auf der Speicherung der gewonnen Daten in Ontologien: Im Speziellen in der Speicherung der Daten in die in der bereichsübergreifenden Projektarbeit erzeugte Ontologie. Wie

[30] Wie bereits in Punkt 5.4 angesprochen, wird die API und nicht der Camunda Modeler verwendet, da die Camunda API in eine Entwicklungsumgebung importiert und so direkt in die Ontologie gespeichert werden kann.

[31] Maven, vom Hersteller Apache, dient der im Software Projekt Management zur Verwaltung von Builds und anderen wichtigen Dokumenten an zentraler Stelle (vgl. [Maven]). So können mittels Maven so genannte „Dependencies“ oder Abhängigkeiten definiert werden, die dann vom Tool selbst geladen und importiert werden. Dadurch wird der Umgang von Abhängigkeiten stark vereinfacht, da nicht mehr jede einzelne Klasse als Java .jar File eigenhändig heruntergeladen und importiert werden muss. In dieser Arbeit wurde Maven als Eclipse Plugin verwendet.

[32] Aus der java eigenen Bibliothek java.util.

auch beim Einlesen von Daten, so basiert auch die Speicherung auf bereits Vorhandenem. Diese Speicherung von BPMN Elementen in der Ontologie basiert auf dem in Punkt 3.4 vorgestelltem Konzept, das in der bereichsübergreifenden Projektarbeit erstellt wurde. Sowohl in der bereichsübergreifenden Projektarbeit als auch in [Wageneder 2017] wurde die Ontologie im Tool Protégé implementiert. Da für dieses Tool eine eigene API angeboten wird, wurde diese als Startpunkt für die Speicherung in der Ontologie verwendet[33].

6.2.1. Protégé API

Wie in 6.2 erwähnt, bietet das Tool Protégé eine eigene API an. Da dies eine direkte Verbindung zu der in [Wageneder 2017] erstellten und in der bereichsübergreifenden Projektarbeit erweiterten Ontologie verspricht, wurde diese API als bevorzugte Variante zur Speicherung der Ontologie gewählt. Wie auch die Camunda API bietet die Protégé API dabei eine java Schnittstelle an. Des Weiteren bietet Protégé Dokumentation über Funktionsweisen in einem Wiki an[34]. Diese Dokumentation bietet eine breite Auswahl an Hilfestellung, von der Installation der Abhängigkeiten bis zur Anwendung verschiedenster Klassen. Um die Abhängigkeiten der Protégé API zu importieren wurde Maven verwendet.

Die Dokumentation von Protége ist zwar hilfreich, doch leider ist die erste Seite des Protégé Wikis beispielsweise zuletzt im Jahre 2010 bearbeitet worden. Ähnlich verhält es sich mit anderen Seiten dieses Wikis oder mit nötigen Abhängigkeiten der Protégé API. Dies erschwert die Arbeit mit der API und da teils sehr veraltete Bibliotheken mittels Maven geladen werden mussten, wurde von diesem Versuch abgegangen.

6.2.2. OWL API

Neben der API, die vom Tool Protégé zur Verfügung gestellt wird, gibt es auch andere Möglichkeiten, um Ontologien, die dem OWL Standard entsprechen via API zu laden, zu bearbeiten und zu speichern. Eine Möglichkeit hierbei stellt die OWL API dar, welche auch in java implementiert ist. Von dieser API wurde auch im Konzept der ereichsübergreifenden Projektarbeit ausgegangen. Neben der Unterstützung des OWL 2 Standards bietet die OWL API mehrere XML Syntax[35] an, in denen Ontologien gespeichert sein können. Auch Reasoner, eine Möglichkeit um logische Konsequenzen verschiedener Verbindungen zu erfassen und zu erfragen,

[33] Obwohl im Konzept die OWL API anstatt der Protégé API vorgeschlagen wurde. Die Protégé API schien aufgrund der Toolwahl der Vorarbeiten als treffendere Wahl.

[34] Vgl. https://protegewiki.stanford.edu/wiki/ProtegeOWL_API_Programmers_Guide (16.05.2018)

[35] In dieser Arbeit wird die Syntax RDF/XML verwendet.

sind in der OWL API inkludiert und können entsprechend angepasst werden[36]. Wie die Protégé API kann auch die OWL API mittels Maven importiert werden.

Als größter Vorteil der OWL API kann die Aktualität gesehen werden. Während die Protégé API viele unterschiedliche Abhängigkeiten benötigt, die teils wie der Maven Protégé Editor[37] aus dem Jahr 2015 kommen, ist dies bei der OWL API nicht der Fall. Um die OWL API und alle Abhängigkeiten zu laden, ist nur eine Maven „Dependency" nötig. Abbildung 10 zeigt die benötigten „Dependencies", die mit der OWL API vorausgesetzt werden. Camunda wird benötigt um BPMN Modelle wie in 6.1 beschrieben zu laden. Um die Ontologie zu laden, zu bearbeiten und zu speichern wird einzig und allein die „net.sourceforge.owlapi" Dependency benötigt.

```
<dependencies>
    <dependency>
        <groupId>org.camunda.bpm.model</groupId>
        <artifactId>camunda-bpmn-model</artifactId>
        <version>7.8.0</version>
    </dependency>
    <dependency>
        <groupId>net.sourceforge.owlapi</groupId>
        <artifactId>owlapi-distribution</artifactId>
        <version>5.1.0</version>
    </dependency>
</dependencies>
```

Abbildung 10: Die in dieser Arbeit mit der OWL API benötigten „Dependencies"

Um Ontologien mit der OWL API zu laden wird ein „OWLOntologyManager" der gleichnamigen Klasse benötigt. Mithilfe eines Ontology Managers können Ontologien geladen werden. Eine Instanz eines Ontologie Managers kann dabei mehrere Ontologien verwalten. Ontologien können dabei über eine URL, ein File oder ein so genanntes IRI[38] geladen werden. Ein weiterer wichtiger Bestandteil der OWL API ist die Klasse „OWLDataFactory", die ebenfalls vom Ontology Manager zur Verfügung gestellt wird. Von dem Ontology Manager und der Data Factory wurde in dieser Arbeit nur je eine Instanz benötigt. Die Data Factory ist besonders wichtig, als das von dieser Klasse verschiedene Axiom[39] und Assertion[40] Klassen zur Verfügung gestellt werden.

[36] Eine Fragestellung für einen Reasoner der Ontologie aus der bereichsübergreifenden Projektarbeit wäre: „Ist Process ein BaseElement?". Obwohl nicht direkt veerbt, liefert ein Reasoner als Antwort: „Process owl:Subclass of CallableElement, CallableElement owl:Subclass of RootElement, RootElement owl:Subclass of BaseElement". Daraus folgt: „Process owl:Subclass of BaseElement."

[37] Vgl. https://mvnrepository.com/artifact/edu.stanford.protege/org.protege.editor.owl (16.05.2018)

[38] IRI steht hierbei für Internation Ressource Identifier und ist eine Untergruppe der URIs (vgl. [IRI RFC3987]).

[39] Der theoretische Hintergrund zu Axiomen wurde bereits in Punkt 2.5 erläutert.

[40] Eine Assertion stellt dabei jeweils die Zugehörigkeit eines Elementes zu einer Klasse dar. Beispielsweise würde eine Assertion „Tisch" zur Klasse „Möbelstück" bedeuten, ein Tisch ist ein Möbelstück. Assertions können nicht nur zwischen Elementklassen bestehen, sondern auch zwischen Instanzen und den dazugehörigen Elementklassen (vgl. [OWL Syntax 2012]).

Die Axiom Klassen, die in dieser Arbeit verwendet wurden sind:

- Declaration Axiom: Ein Declaration Axiom wird gebildet, um Elemente in der Ontologie zu speichern. Durch die Deklarierung eines Elementes ist zwar der Name dieses Elementes definiert, allerdings müssen noch alle Attribute gesetzt werden.
- Class Assertion Axiom: Diese Klasse ist eine Mischung aus Axiom und Assertion. Hierbei wird ein Axiom verwendet, um die Zugehörigkeit eines Elementes zu einer Klasse darzustellen.
- Object Property Assertion Axiom: Bei einem Object Property Assertion Axiom[41] wird die Zugehörigkeit eines Elementes als Attribut eines anderen Elementes dargestellt.
- Data Property Assertion Axiom: Ein Data Property Assertion Axiom ist einem Object Property Assertion Axiom sehr ähnlich[42]. Allerdings wird hierbei die Zugehörigkeit eines Wertes, egal welchen Datentypes, eines Attributes eines Elementes dargestellt.

Ein weiteres wichtiges Element der API ist die Klasse „OWLNamedIndividual“. Named Indiviuals werden im OWL Standard verwendet, um ein spezifisches Element anzusprechen. Diese spezifischen Elemente werden mittels einer eindeutigen IRI referenziert.

6.3. Vorgehensweise bei der Speicherung von BPMN Modellen in Ontologien

In Punkt 6.1 wurde beschrieben, wie BPMN Modelle mittels der Camunda API eingelesen werden können. Dadurch sind Collections aller relevanten BPMN Elemente bereits vorhanden. Wie in Punkt 6.2.2 beschrieben ist es mithilfe der Klasse „OWLOntologyManager“ der OWL API möglich, Ontologien, wie die aus der bereichsübergreifenden Projektarbeit entstandene Ontologie, zu laden. Um alle Elemente der BPMN Modelle in der Ontologie zu speichern, müssen die Collections Element für Element durchgegangen werden. Die Elemente sind dabei als Typ „ModelElementInstance“ gespeichert, und müssen noch auf den richtigen Typ, also beispielsweise Task gecasted[43] werden. Erst dadurch können die Typen spezifischen Methoden wie „getId()“ angewendet werden.

[41] Um Object Property Assertion Axiome zu erstellen wird eine Object Property Expression benötigt. Diese wird erstellt aus der IRI der Ontologie sowie dem Namen der jeweiligen Object Property. Object Properties beginnen in der Ontologie der bereichsübergreifenden Projektarbeit jeweils mit dem Verb „has”.

[42] Wie Object Property Assertion Axiome benötigen Data Property Assertion Axiome Expressions. Diese sind gleich aufgebaut wie Object Property Expressions.

[43] Obwohl casts als schlechte Praxis gelten, stellen sie hier kein Problem dar und sind alternativlos. Da jede Collection mithilfe der Camunda API genau einen Elementtyp enthalten, können casts problemlos durchgeführt werden.

Named Individuals funktionieren ähnlich wie Declartion Axiome. Allerdings werden Named Individuals nicht als OWL Klasse, sondern als Entity gespeichert. Dennoch benötigen Named Individuals wie OWL Klassen eindeutige IRIs.

Bei Class Assertion Axiomen wird die Verbindung der Elemente zu den Klassen mittels der BPMN Typennamen gewährleistet.

Für Object Property und Data Property Assertion Axiome müssen jeweils zuerst die Property Expressions gebildet werden. Bei Data Property Assertion Axiome werden anschließend Datenwerte mit den jeweiligen Individuals und Expressions übergeben. Bei Object Property Assertion Axiomen werden die Expression, das Individual dessen Attribut gesetzt und das referenzierte Individual übergeben.

Um Axiome zur Ontologie hinzuzufügen gibt es die Methode „add“ der Ontologie. Zum Schluss muss die Ontologie wieder gespeichert werden, in dieser Arbeit wird die Ontologie mit Hilfe des Ontologie Managers im RDF/XML Format wiederum in einem File gespeichert.

7. Proof of Concept

In diesem Kapitel wird das in Kapitel 6 vorgestellte Konzept analysiert, auf Rahmenbedingungen hingewiesen sowie die Funktion geprüft und reflektiert.

7.1. Zusammenfassung der verwendeten Software

Das Konzept wurde in „Eclipse IDE for Java Developers“ implementiert. Verwendet wurde dabei Version Oxygen.3 Release (4.7.3). Wie bereits in den Punkten 6.1 und 6.2.2 beschrieben, wurde um BPMN Modelle zu laden die Camunda API in Version 7.8.0 verwendet. Die OWL API, um die Ontologie zu laden und in die Ontologie zu speichern, in Version 5.1.0. Um die beiden APIs in Eclipse zu verwenden, wurde Maven als Eclipse Plugin „Maven Integration for Eclipse (Luna)“ in der Version 1.5.0 angewendet. Als Java Version wurde „Java(TM) SE Runtime Environment (build 9.0.4+11)“ benutzt. Zur Veranschaulichung der Ontologie wurde Protégé in der Version 5.2.0 verwendet.

7.2. Implementierte BPMN Elemente

Um das Konzept zu prüfen wurden beschreibende Elemente[44] aus dem Standard BPMN angewendet (vgl. BPMN [2014]).

Der Standard gibt dabei nicht nur Elemente sondern auch Attribute vor. Während die im Standard definierten Elemente verwendet wurden, wurden die Attribute laut Vorgabe der Ontologie aus der bereichsübergreifenden Projektarbeit implementiert. Daraus ergebende Probleme sind unter Punkt 7.3 beschrieben.

7.3. Die Ontologie aus der bereichsübergreifenden Projektarbeit als Basis des Konzepts

Die im Punkt 3.3 vorgestellte Ontologie aus der bereichsübergreifenden Projektarbeit wurde als Referenzontologie des Standards BPMN verwendet. In der Implementierung treten dabei Probleme[45] auf. Die folgenden Unterpunkte beschreiben die Probleme mit der Ontologie sowie eine Reflexion des Inhaltes.

7.3.1. Attribute von Instanzen

So wurden in der Ontologie aus der bereichsübergreifenden Projektarbeit Attribute von Instanzen einer Klasse als Pflichtattribute inkludiert. Attribute eines „UserTasks“ wie „actualOwner“ oder „taskPriority“ sind, obwohl Pflichtattribute, im Rahmen der

[44] Beschreibende Elemente werden auch als Konformitätsklasse 1 bezeichnet.

[45] Die Reflexion der Ontologie erfolgte nur für beschreibende Elemente sowie Superklassen dieser Elemente.

Modellierung nicht bekannt[46]. Im Gegensatz zu „state“, auch ein Instanz Attribut, besitzen diese aber laut Standard keinen Default Wert. Für das Konzept wurde daher für „actualOwner“ der Wert „None“[47] und für „taskPriority“ der Wert 1 verwendet.

7.3.2. Wertlose Attribute und Probleme durch Vererbung

Die Ontologie aus der bereichsübergreifenden Projektarbeit ist sehr stark auf Vererbung und Mehrfachvererbung ausgelegt. Dies ist sinnvoll für die Struktur, führt aber für die Speicherung von Elementen zu Problemen und sinnlosen Attributen. Beispielsweise erbt das „StartEvent“ durch die Klasse „Event“ von der Klasse „FlowNode“ die Attribute „hasIncoming“ und „hasOutgoing“ vom Typ „SequenceFlow“. Diese Attribute müssen nicht gesetzt sein (mindestens null), dennoch wäre ein „StartEvent“ mit einem eintreffenden „SequenceFlow“ ein Widerspruch und das Attribut ist so wertlos für dieses Element. Vererbte Attribute dieser Art sind zwar kein Widerspruch an sich, da optional, aber führen zu einer langen Liste an Attributen und so zu erhöhter Fehleranfälligkeit in der Implementierung.

Weitere Probleme durch die Vererbung, vor allem die „id“ betreffend, werden im Punkt 7.5.2 beschrieben.

7.3.3. Probleme durch zweiseitige Beziehungen

Attribute von BPMN Elementen werden in den Tabellen der jeweiligen Elemente bestimmt. In manchen Fällen werden Beziehungen, die laut der Tabellen einseitig sind, in einem UML Diagramm beidseitig dargestellt. Ein Beispiel hierfür wäre die Beziehung zwischen „BaseElement“ und „Association“.

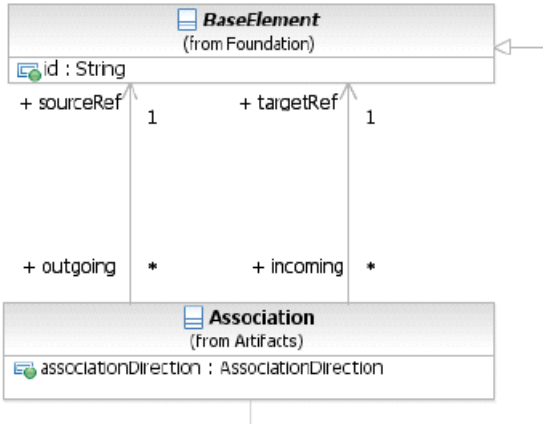

Abbildung 11: Beziehung „BaseElement“ zu „Association“ (entnommen aus [BPMN 2014], S.65)

In Abbildung 11 sieht man die Beziehung laut dem UML Diagramm. Die Beziehungen von „Association“ in Richtung „BaseElement“ („targetRef“ und „sourceRef“) sind

[46] Deshalb stellen diese Attribute ein Problem für die Validierung dar, weil sie verpflichtend aber im Rahmen der Modellierung nicht bekannt sind, beziehungsweise bekannt sein müssen.

[47] Wie das Attribut „state“.

auch in der Tabelle von „Association“. Umgekehrt sind die Beziehungen zwar auch benannt, aber nicht in der Tabelle von „BaseElement“ vorhanden. In der Ontologie aus der bereichsübergreifenden Projektarbeit wurden diese Beziehungen auch erstellt. Die beiden Attribute „hasIncoming“ und „hasOutgoing“ des Typs „Association“ werden weitervererbt. Dadurch sind „hasIncoming“ und „hasOutgoing“ beispielsweise in der Klasse „FlowNode“ nicht eindeutig. Diese hat ebenfalls die Attribute „hasIncoming“ und „hasOutgoing“ aber vom Typ „SequenceFlow“. „FlowNode“ erbt indirekt[48] von „BaseElement“. Viele Klassen wie „Event“ oder „Activity“ haben so jeweils zweimal die Attribute „hasIncoming“ und „hasOutgoing“, aber jeweils mit einem anderen Typ.

7.3.4. Nötige Änderungen und Reflexion

Die einzige Änderung an der Ontologie aus der bereichsübergreifenden Projektarbeit ist die in Punkt 7.3.3 beschriebene nicht eindeutige Verwendung von „hasIncoming“ und „hasOutgoing“. Da diese Beziehung der Klasse „BaseElement“ mit der Klasse „Association“ nicht in der Tabelle von „BaseElement“ verzeichnet ist, wird auf diese verzichtet und sie aus der Ontologie entfernt. Die Probleme mit Instanz Attributen konnte mittels definierten Default Werten ohne Änderung der Ontologie behoben werden.

Grundsätzlich ist die Ontologie für die Speicherung von BPMN Modellen geeignet. Wie in den zuvor erwähnten Punkten kommt es aber zu Konflikten. Diese beruhen darauf, dass die Entwicklung der Ontologie aus dem Standard erfolgte. Dadurch wurden Annahmen, die die Implementierung der Speicherung erleichtern könnten, nicht getroffen.

Es wurden nur die beschreibenden Elemente des BPMN Standards und die Superklassen dieser untersucht, es können folglich noch Änderungen anderer Elemente nötig sein.

7.4. Einschränkungen

Das Konzept wurde nur mit den in Punkt 7.1 erläuterten Versionen getestet. Durch die ausschließliche Verwendung von beschreibenden Elementen ergeben sich zusätzliche Einschränkungen. Alle verpflichtenden und optionalen Attribute der Elemente wurden implementiert, sofern sie nicht Elemente anderer Konformitätsklassen als Attributtyp besitzen. Des Weiteren wurde auf offensichtlich irrelevante vererbte Attribute verzichtet. Ein Beispiel hierfür wäre „hasIncoming“ der Klasse „StartEvent“ wie in Punkt 7.3.2 beschrieben.

Aufgrund der fehlenden Funktion in der Camunda API können „Group“ Elemente, die in der Konformitätsklasse 1 enthalten sind, nicht erfasst werden. Da „Group“ Elemente allerdings als einzig eigenes Attribut „hasCategoryRef“ vom Typ „CategoryValue“ besitzen, werden diese Elemente verwendet um Groups anzusprechen. Laut

[48] Über die Klasse „FlowElement“

[BPMN 2014] werden „Group“ Elemente verwendet um Elemente mithilfe von „CategoryValue“ zu gruppieren. In dem hier vorgestellten Konzept werden „CategoryValue“ Elemente mit dazugehörigen „Group“ Elementen erstellt. Die „id“ für das „Group“ Element wird dabei die „id“ des dazugehörigen „CategoryValue“ Elements mit dem Zusatz „-group“.

Bei einigen vererbten Attributen von Elementen gibt es Probleme. Während bei einzelnen Attributen über die Methode „getValue“ mit dem jeweiligen Attributnamen auf einen Wert zugegriffen werden kann, geht das bei Attributen mit mehreren Werten nicht. Beispielsweise besitzt die Klasse „CallActivity“ das Attribut „hasLanes“ mit null oder mehreren Elementen der Klasse „Lane“. Dieses Attribut ist in der Camunda API, da nur vererbt, nicht implementiert. Dadurch kann dieses Attribut für „CallActivity“ nicht verwendet werden.

7.5. Testen des Konzepts

Während das Einlesen der BPMN durch die Camunda API kontrolliert wird und auf Fehler wie falsche Attribute hingewiesen wird, gibt es bei der OWL API keine solchen Mechanismen. Um die Korrektheit des Konzepts zu erproben werden drei unterschiedliche Beispielprozesse dreier unterschiedlicher Herausgeber mittels vier unterschiedlichen Tools getestet. Dies soll nicht nur die beliebige Verwendbarkeit des Konzeptes, sondern auch die unterschiedlichen Anwendungen des BPMN Standards zeigen.

7.5.1. Vorgehensweise

Als BPMN Tools um den Prozess zu testen wurden verwendet:

- BPMN2-Modeler Plugin: Dieser Modeler wurde bereits in Punkt 4.1 beschrieben. Verwendet wurde Version 1.4.2
- Camunda Modeler: Camunda Modeler wurde in 4.2 beschrieben. Verwendet wurde Version 1.14.0
- Signavio Process Manager: Signavio ist ein Webbasiertes Tool. Mit Signavio können BPMN Prozesse im Webbrowser modelliert und validiert werden. Außerdem können Prozesse importiert und exportiert werden[49].
- Bizagi: Bizagi bietet Software von der Modellierung bis zur Ausführung von Prozessen an. Mit Bizagi Modeler können modellierte Prozesse unmittelbar ausgeführt und getestet werden[50]. Verwendet wurde Version 3.2.6.094

[49] Vgl. https://www.signavio.com (16.05.2018). Signavio bietet eine Academic Version für Hochschulen an, welche in diesem Fall verwendet wurde.

[50] Vgl. https://www.bizagi.com/en/products/bpm-suite/modeler (16.05.2018)

Diese Tools decken ein sehr breites Spektrum ab: Ein Tool, das als Plugin verwendet wird (BPMN2-Modeler), ein Open Source Tool (Camunda Modeler), ein Webplugin (Signavio) und ein Tool für Enterprise Anwendung ohne Open Source (Bizagi).

Als Beispielprozesse wurden verwendet:

- Shipment Process of a Hardware Retailer: Ein Prozess, den auch die OMG als Beispiel für BPMN im auf der OMG Website verlinkten Tutorial[51] aufweist.
- Teile beschaffen: Dieser Prozess wird als Beispiel von Signavio in der Academic Version zur Verfügung gestellt.
- Invoice: Invoice ist ein Beispielprozess von Camunda[52].

Beispielprozesse von Bizagi und BPMN2-Modeler Plugin konnten nicht gefunden werden.

Um die Prozesse in unterschiedlichen Versionen zu bekommen, wurde versucht die jeweiligen Prozesse zu importieren und anschließend zu exportieren. Dabei hat sich herausgestellt, dass zum einen die jeweiligen Files in Camunda und im BPMN2-Modeler nicht verändert werden. Zum anderen war es zwar mit Bizagi möglich alle Prozesse zu importieren und so darzustellen, allerdings nicht mehr diese zu exportieren. Nur der Prozess „Teile beschaffen“ konnte erneut exportiert werden.

Obwohl das BPMN2-Modeler Plugin für Eclipse alle Prozesse importieren kann und diese nicht verändert, treten Fehler auf. Abbildung 12 zeigt den von Eclipse angezeigten Fehler, nach der Importierung des Prozesses „Shipment Process of a Hardware Retailer.“

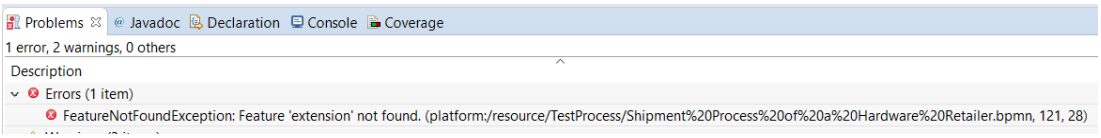

Abbildung 12: Fehler in BPMN2-Modeler Plugin

7.5.2. Probleme

Während des Testens traten Probleme mit der „id“ der „Documentation“ Klasse auf. Da „Documentation“ von „BaseElement“ erbt, müsste das Attribut „id“ immer verpflichtend gesetzt sein. Auftretende Probleme liegen an der Formulierung im Standard. Abbildung 13 zeigt, wie das Attribut „id“ des Elements „BaseElement“ im Standard beschrieben ist.

[51] Vgl. http://www.omg.org/news/meetings/workshops/HC-Australia/Mancarella.pdf (12.05.2018)

[52] Vgl. https://github.com/camunda/camunda-bpm-platform/blob/master/examples/invoice/src/main/resources/invoice.v1.bpmn (16.05.2018)

Table 8.5 – BaseElement attributes and model associations

Attribute Name	Description/Usage
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.

Abbildung 13: „BaseElement“ Attribut „id" (entnommen aus [BPMN 2014], S.54)

So ist eine „id“ zwar verpflichtend, es kann aber, sollte ein Element nicht referenziert werden, auf eine „id“ verzichtet werden. Somit kann auf eine eindeutige ID nicht mit Sicherheit zugegriffen werden. Dies wird aber für eine eindeutige Zuordnung benötigt. Obwohl in den meisten Fällen das Attribut „id“ gesetzt sein wird, kann davon nicht mit Sicherheit ausgegangen werden.

Sollten Elemente nicht zu den beschreibenden Elementen gehören, werden diese von der Camunda API nicht verwendet. Es kann aber, im Falle das das Element via „SequenceFlow“, „Association“, oder „MessageFlow“ als „hasSourceRef“ oder „hasTargetRef“ referenziert wird, als ID ohne Typzuweisung in der Ontologie gespeichert sein. Fehler treten dabei nicht auf.

7.5.3. Ergebnis

Abbildung 14 zeigt den aus dem OMG BPMN Tutorial entnommenen Prozess, der das Verschicken von Artikeln aller Art darstellt.

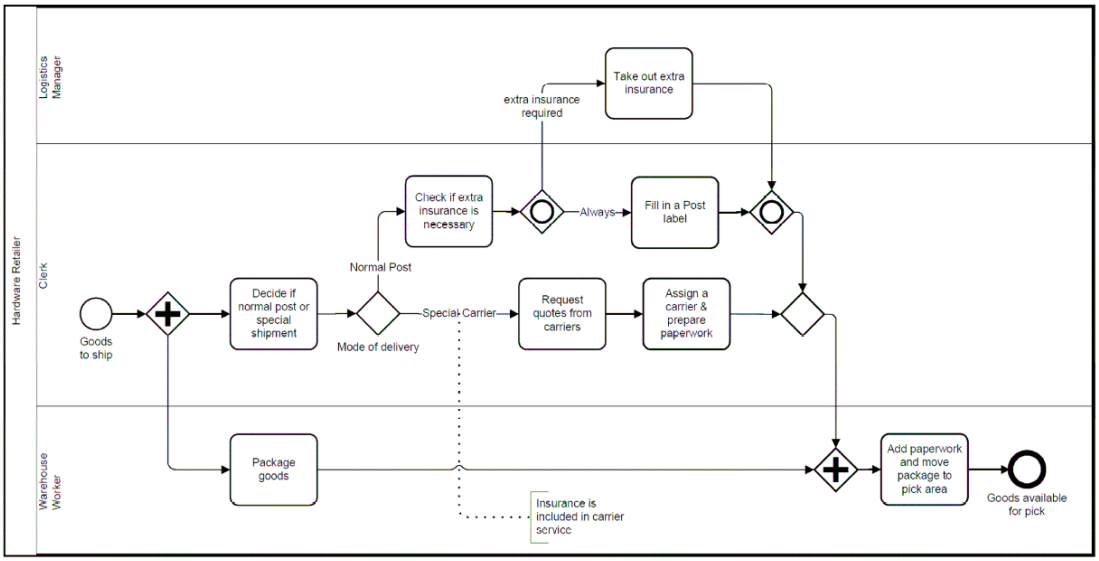

Abbildung 14: Prozess aus OMG BPMN Tutorial

Um die Speicherung in der Ontologie aus der Bereichsübergreifenden Projektarbeit beispielhaft zu zeigen, weist Abbildung 15 alle „Tasks“ aus dem in Abbildung 14 gezeigten Prozess. Verwendet wurde hierfür der weder importierte noch exportierte Prozess.

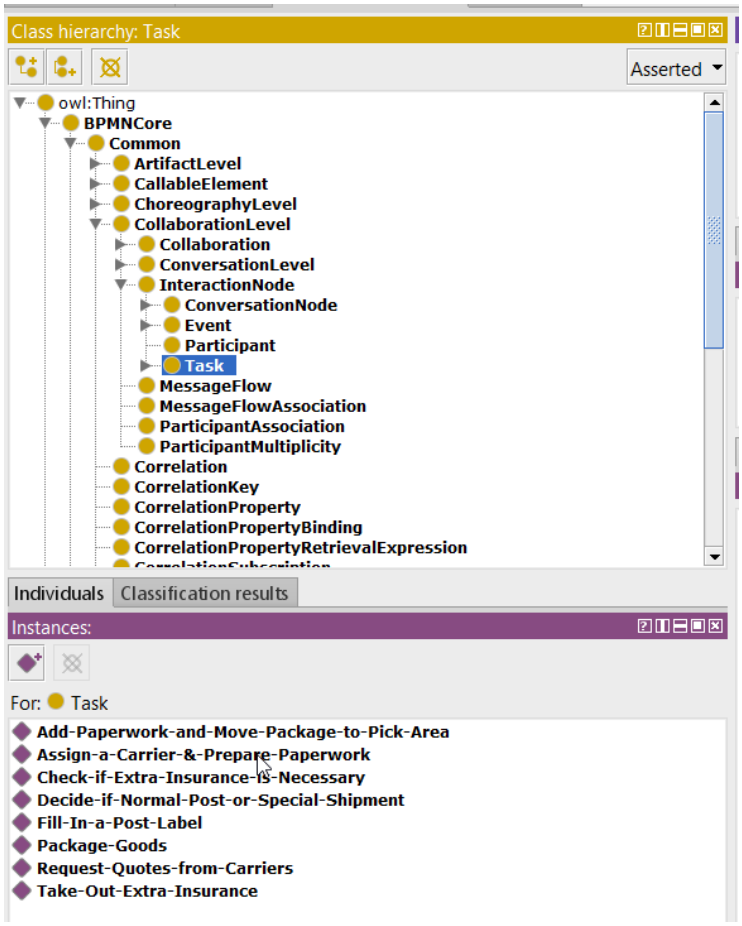

Abbildung 15: Instanzen von Task in der Ontologie

Abbildung 16 zeigt alle Attribute mit den entsprechenden Werten für den Task „Add-Paperwork-and-Move-Package-to-Pick-Area“.

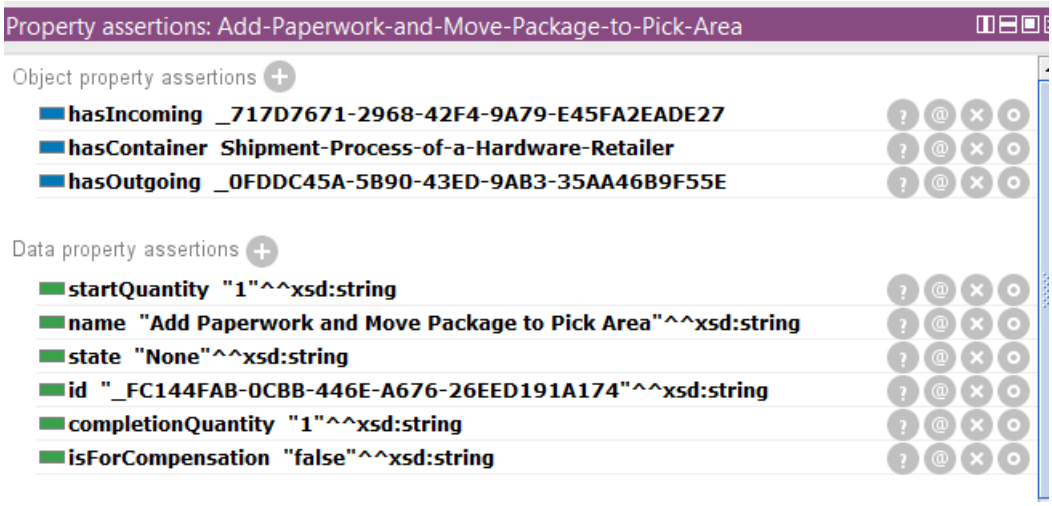

Abbildung 16: Attribute von „Add-Paperwork-and-Move-Package-to-Pick-Area"

Abbildung 17 zeigt die Attribute des gleichen Tasks, allerdings wurde der Prozess zuvor mit Signavio importiert und exportiert. Die Attribute in diesem Fall sind gleich.

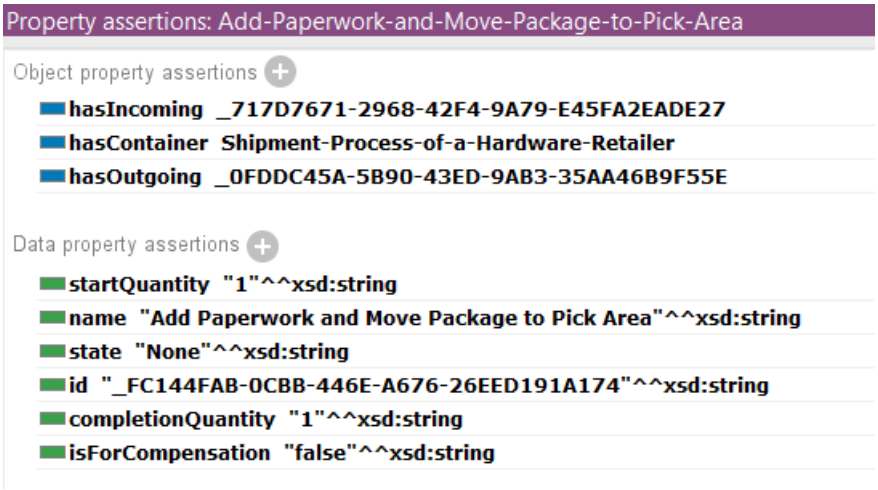

Abbildung 17: Attribute nach Import und Export mit Signavio

Jedoch enthält der Prozess nach dem Import und Export mittels Signavio im Gegensatz zu davor ein Element des Typs „Participant“.

Diese Tests haben gezeigt, dass das im Konzept beschriebene Vorgehen funktioniert. Beliebig gewählte Prozesse aus einer Mehrzahl an Editoren können in einer Ontologie gespeichert werden. Probleme können entstehen durch fehlende „id“ Attribute von Elementen. Diese Fehler können durch das Konzept nicht aufgefangen werden, da eine eindeutige Zuordnung entweder durch den Namen oder durch einen Identifier nötig ist. Daher wird das Konzept durch Fehlermeldungen so angepasst, dass das Element mit fehlendem Identifier leicht identifiziert werden kann. Das Attribut „id“ muss dann im Editor gesetzt werden.

7.6. Reflexion

Prozesse des Standards BPMN können in der Ontologie, die in der bereichsübergreifenden Projektarbeit erstellt wurde gespeichert werden. Dies ist allerdings nur mit Einschränkungen der Fall. Gerade die zuvor erwähnte fehlende Eindeutigkeit von Elementen durch fehlende Identifier sind ein großes Problem. Dadurch zeigt sich ein Vorteil der Formalisierung des Standards. Ein Attribut kann in einer Ontologie entweder verpflichtend sein oder optional. Die Formulierung beim Attribut „id“ für „BaseElement“ ist ein sehr gutes Beispiel, wie Ontologien den Standard und die Konformitäten verbessern können.

Die Speicherung von BPMN Modellen in dieser Ontologie ist hierbei die Basis, um BPMN Modelle mithilfe der Ontologie aus der Bereichsübergreifenden Bachelorarbeit zu validieren. Dadurch entstehen BPMN Modelle, die dem formalisierten Standard entsprechen und führen so zu einer erhöhten Modellqualität[53].

[53] Dies bezieht sich nicht auf modellierte Prozesse, aber auf die Konformität des Modells zum Standard.

Des Weiteren waren zwar nur in selten Fällen Inkompatibilitäten festzustellen[54], dennoch können Prozesse nach Import und Export verändert sein. Obwohl diese im angezeigten Diagramm optisch noch gleich sind, hat sich das Modell verändert. Dies wird in der Ontologie mit zusätzlichen oder veränderten Attributen dargestellt.

Obwohl also jeder Editor auf Standardkonformität prüft, sind unterschiedene Verständnisse des Standards vorhanden. Gerade bei vererbten Attributen ist dies, wie in 7.4 und 7.5.2 beschrieben, ein Problem.

Verändert sich die Ontologie, die geladen und in die gespeichert wird, kann das zu unerwarteten Effekten führen. Bei Klassen der Ontologie[55] ist dies nicht der Fall. Ist eine Klasse, die geladen werden soll, nicht vorhanden, wird ein Fehler produziert und die Ontologie nicht verändert. Die Klasse muss in der Ontologie erstellt oder im Prozessmodell an die vorhandene Klasse der Ontologie angepasst werden. Findet die OWL API kein Äquivalent einer Object Property wie etwa „hasContainer“, wird diese neu erstellt[56].

Werden Pflichtfelder eines Elementes von einem Editor nicht gesetzt, so tritt ein Fehler in der Prozedur auf. Diese wird abgebrochen und eine Fehlermeldung mit der Funktion, in der es Probleme gab, angezeigt. Dadurch kann erkannt werden mit welchem Elementtyp Probleme auftraten.

[54] Die Ausnahme hierbei ist Bizagi.

[55] Beispielsweise „BaseElement“.

[56] Dies kann zu unerwarteten Attributen von Elementen führen.

8. Zusammenfassung

Der Standard BPMN ist eine beliebte Möglichkeit um Prozesse zu modellieren und anschließend ausführbar zu machen. Viele Hersteller bieten dieses Vorgehen an und halten sich dabei strikt an den Standard. Der Standard BPMN ist in einem Fließtext Dokument sowie darin enthaltenen UML Diagrammen definiert. In diesem Dokument stellen sich mehrere Probleme. Zum einen sind Definitionen im Fließtext nicht eindeutig und ohne Interpretationsspielraum. Des Weiteren sind zwischen dem Fließtext und den UML Diagrammen Widersprüche erkennbar.

Dies führt zu Interpretationsspielraum für Hersteller von BPMN Editoren und so zu Inkompatibilitäten zwischen Modellen einzelner Editoren. Obwohl es Initiativen zur Beseitigung solcher Inkompatibilitäten gibt, können diese weder ganz ausgeschlossen noch zu 100% vermieden werden.

Ontologien sind eine gute Möglichkeit um Interpretationsspielräume zu vermeiden und Standards wie BPMN zu formalisieren. So wurde im Rahmen der bereichsübergreifenden Projektarbeit eine Ontologie über den BPMN Standard erstellt. Dabei wurde ein generelles Vorgehen bei Konflikten oder Unklarheiten im Standard gewählt.

Um nicht nur den Standard an sich zu formalisieren, sondern auch Modelle Standardkonform zu erstellen müssen diese mithilfe der Ontologie aus der bereichsübergreifenden Projektarbeit validiert werden. Dazu müssen diese Modelle als erster Schritt in der Ontologie gespeichert werden. Da nicht standardkonforme Modelle nicht erstellt werden sollten, empfiehlt sich eine Validierung der Modelle schon während des Modellierens.

Damit BPMN Modelle schon während des Modellierens auf Standardkonformität mittels Ontologie getestet und validiert werden können, müssen diese während der Modellierung in der Ontologie gespeichert werden.

Diese Arbeit beschäftigt sich mit der Frage, ob BPMN Modelle während der Modellierung in der Ontologie der Bereichsübergreifenden Projektarbeit gespeichert werden können. Aufgrund der mangelnden Implementation in dem BPMN2-Modeler Plugins und dem Rahmen dieser Arbeit muss diese Frage mit nein beantwortet werden.

Dennoch wurde in dieser Arbeit ein Vorgehen entwickelt, wie bereits bestehende BPMN Modelle eingelesen und dann in die Ontologie gespeichert werden können. Dazu wurden bereits vorhandene Schnittstellen verwendet. Diese Schnittstellen sind in java implementiert. Aus diesem Grund wurde auch die Konvertierung von BPMN Modellen in die Ontologie in java implementiert.

Dieses Vorgehen wurde mittels definierter Prozesse unterschiedlicher Herausgeber getestet. Implementiert beziehungsweise verändert wurden die Prozesse mit BPMN Editoren unterschiedlicher Hersteller. Hierbei zeigte sich, dass eine Formalisierung des Standards nach wie vor notwendig ist. Gerade Probleme bei eindeutigen Identi-

fiern stellen hier Schwierigkeiten dar. Da eine Ontologie auf einer eindeutigen Zuweisung von Elementen beruht, kann es zu Fehlverhalten kommen, wenn ein Element aus einem BPMN Prozess keine eindeutigen Eigenschaften besitzt. Dies kann auch durch das in dieser Arbeit erstellte Vorgehen nicht ausgeschlossen werden. Dieses Fehlverhalten kann nicht vermieden, aber durch eindeutige Fehlermeldungen an den Anwender kommuniziert werden.

Das der Fehler der eindeutigen Identifier an einer nicht eindeutigen Formulierung im Standard BPMN liegt, zeigt die Wichtigkeit einer weiteren Formalisierung des Standards und von Modellen des Standards. Dies kann durch die Validierung von BPMN Modellen mittels der Ontologie aus der Bereichsübergreifenden Arbeit erreicht werden. Dafür stellt die Speicherung der BPMN Modelle in der Ontologie eine Voraussetzung dar.

9. Literaturverzeichnis

[BPMN 2014]
Object Management Group. Business Process Modelling Language Version 2.0.2. http://www.omg.org/spec/BPMN/2.0.2/ (16.05.2018)

[Bucher, Winter 2009]
Bucher, T., Winter, R.: Geschäftsprozessmanagement –Einsatz, Weiterentwicklung und Anpassungsmöglichkeiten aus Methodiksicht. In: HMD Praxis der Wirtschaftsinformatik. Volume 46. Issue 2. Springer. 2009

[Di Martino et al 2016]
Di Martino, B., Esposito, A., Maisto, S. A., Nacchia, S.: A methodology and implementing tool for semantic business process annotation. In: International Workshop on Business Process Modeling, Development and Support. Springer International Publishing. 2016. S. 80-94

[Gupta 2017]
Gupta, G.: jBPMN Suite. NetBeans Plugin. GitHub.
https://github.com/jGauravGupta/jBPMNSuite (16.05.2018)

[Hofmann 2015]
Hoffmann, D.: Theoretische Informatik. Carl Hanser Verlag. 3.Auflage. 2015

[Hofstrand, Holz-Clause 2009]
Hofstrand, D., Holz-Clause, M.: What is a Feasibility Study?. IOWA State University. Ag Decision Maker. Extension and Outreach. https://www.extension.iastate.edu/agdm/wholefarm/html/c5-65.html (16.05.2018)

[IRI RFC3987]
IRI. RFC3897. https://tools.ietf.org/html/rfc3987 (16.05.2018)

[Mädche et al 2001]
Mädche, A., Staab, S., Studer, R.: Ontologien. Wirtschaftsinformatik. 43(4). 2001. S. 393-395

[Maven]
Maven. Apache. https://maven.apache.org/ (16.05.2018)

[OWL 2012]
OWL 2 Web Ontology Language. W3C OWL Working Group. Version 2. https://www.w3.org/TR/owl2-overview/ (16.05.2018)

[OWL Syntax 2012]
OWL 2 Web Ontology Language Syntax. W3C OWL Working Group. Version 2. https://www.w3.org/TR/owl2-syntax (16.05.2018)

[Poetzsch-Heffter 2009]
Poetsch-Heffter, A.: Konzepte objektorientierter Programmierung. Mit einer Einführung in Java. Springer. 2009

[Red Hat]
Red Hat Inc. jBPM. https://www.jbpm.org/ (16.05.2018)

[Rospocher et al 2014]
Rospocher, M., Ghidini, C., Serafini, L.: An ontology for the Business Process Modelling Notation. Formal Ontology in Information Systems - Proceedings of the Eighth International Conference. FOIS2014. September. 22-25. 2014. Rio de Janeiro. Brazil. vol. 267. IOS Press. 2014. S. 133-146,

[Stanford 2016]
Protégé. Stanford Center for Biomedical Informatics Research. Stanford University. 2016. https://protege.stanford.edu/ (16.05.2018)

[Studer 2016]
Studer, R.: Ontologien. Enzyklpädie der Wirtschaftsinformatik. Universität Potsdam. 2016. http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/daten-wissen/Wissensmanagement/Wissensmodellierung/Wissensreprasentation/Semantisches-Netz/Ontologien/index.html (16.05.2018)

[Uschold, Gruninger 2004]
Uschold, M., Gruninger, M.: Ontologies and semantics for seamless connectivity. SIGMOD Rec. 33. 4. 2014. S. 58-64.

[Wageneder 2017]
Wageneder, M.: Ontologische Analyse von BPMN 2.0. Bachelorarbeit 2. FH Joanneum. 2017