Bachelorarbeit

Action Item Ontology

Rayner Vesely

13. August 2014

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

Aufgabenbeschreibung

Ziel dieser Bachelorarbeit ist es, ein semantisches Modell für eine Semantic Web Applikation für Maßnahmen (Action Items) zu entwerfen, das es ermöglicht

- Action Items aus anderen Systemen zu importieren
- Action Items in einer Semantic Web konformen Datenbank zu speichern
- Action Items mittels Semantic Web konformen Abfragesprachen (bspw. SPARQL) zu analysieren
- Action Items anderen Systemen bereitzustellen

Für diesen Datenaustausch soll eine geeignete Ontologie entwickelt werden, die es ermöglicht beliebig attributierte Action Items aus anderen Systemen auszudrücken.

Hierzu müssen im Detail folgende Teilaufgaben bearbeitet werden:

- Anforderungsanalyse an repräsentativen Systemen, die Action Items verwalten
- Modellierung einer Ontologie für Action Items unter Berücksichtigung oben genannter Datenaustauscherfordernisse
- Entwicklung einer Architektur für ein Semantic Web basiertes Action Item Applikation, die oben beschriebene Fähigkeiten aufweist.
- Entwicklung einer geeigneten Evaluations- und Nachweisstrategie (QS)
- Vergleichende Betrachtung und Auswahl geeigneter Semantic Web konformer Datenformate und Datenbanken zur Ontologiemodellierung und Datenspeicherung
- Implementierung inkl. Test/Nachweis der Action Item Applikation
- Begleitende Dokumentation der oben angegebenen Schritte, Designentscheidungen und Ergebnisse (Bachelorarbeit)

Erklärung zur Bachelorarbeit

Ich versichere, dass ich die Arbeit selbständig, ohne fremde Hilfe verfasst habe. Bei der Abfassung der Arbeit sind nur die angegebenen Quellen benutzt worden. Wörtlich oder dem Sinne nach entnommene Stellen sind als solche gekennzeichnet. Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, insbesondere dass die Arbeit Dritten zur Einsichtnahme vorgelegt oder Kopien der Arbeit zur Weitergabe an Dritte angefertigt werden.

Bad Schwartau, 13. August 2014 Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Jede Applikation, die von externen Systemen angesprochen werden soll, muss eine Schnittstelle zur Verfügung stellen, die leicht bedienbar und möglichst flexibel ist. Dies ist nahezu unmöglich, da mit jedem externen System neue Anforderungen gestellt werden. Die Entwickler der Applikation müssen die Schnittstelle deshalb ständig erweitern und pflegen, was zur Folge hat, dass diese unaufhörlich wächst, bis man an einem Punkt angelangt ist, an dem nur noch Experten mit dieser Schnittstelle arbeiten können. Nicht nur, dass es kompliziert wird diese zu benutzen, auch müssen die Entwickler eine umfangreiche Dokumentation erstellen und den Programmcode pflegen, welches durch die Größe immer schwieriger wird. Um diesem entgegen zu wirken, müssen die Daten maschinenlesbar aufbereitet werden, sodass diese durch nicht vorher definierte Anfragen durch Menschen abfragbar sind. Die Lösung hierfür könnte das Semantische Web (siehe Abschnitt 2.4) sein.

Ziel dieser Bachelorarbeit ist es, für die Maßnahmenmanagmentapplikation (siehe Abschnitt 2.2.1) PLATO e1ns.actions, folgend nur noch als e1ns.actions genannt, ein semantisches Modell zu entwerfen und eine technische Lösung zu entwickeln, die sie mit anderen semantischen Systemen verbinden kann. Als erster Schritt soll das Modell für Export und Import im Maßnahmenmanagementsystem e1ns.actions erstellt werden. In Anlehnung an andere Standardentwicklung im Semantischen Web wie z.B. Ontologien (siehe Abschnitt 4.3.1) im Gebiet der Mengen und Maßeinheiten oder der Herstellung, wird somit eine Ontologie für Maßnahmen (siehe Abschnitt 2.2.2.1) entwickelt werden. Mögliche technische Lösungen müssen analysiert und eine Auswahl einer möglichen Technik vorgenommen werden. Mit dieser kann dann die Ontologie für Maßnahmen modelliert werden. In PLATO SCIO^{TM} -Methods (siehe Abschnitt 2.1.2) werden alle Daten auch als Triples gespeichert. Dies liegt daran, dass man in der Anwendung frei Attribute definieren kann, mit denen das Basisdatenmodell beliebig erweitert werden kann. Um Abfragen über die erweiterten Graphen (siehe Abschnitt 2.4) vornehmen zu können, müssen die Daten in einem Triplestore (siehe Abschnitt 4.2) vorliegen und über die Graph basierte Abfragesprache SPARQL[1] ansprechbar sein. RDF [2] ist auch das Ex- und Import-Format für e1ns.methods. An diese Architektur soll e1ns.actions angeschlossen werden. Dazu bedarf es einen Ex- und Importformates, mit dem Projekte zwischen verschiedenen Instanzen und Installationen ausgetauscht werden können. Gleichzeitig soll e1ns.actions als ein Provider in einem Firmenverbund als Endpoint für SPARQL-Queries zur Verfügung stehen und mit anderen Ontologien vernetzt werden können.

[1] SPARQL Protocol And RDF Query Language
[2] Resource Description Framework - Modell zur Beschreibung von Ressourcen

2 Grundlagen

In diesem Kapitel werden einige Grundlagen für das bessere Verständnis dieser Bachelorarbeit aufgeführt und erläutert.

2.1 Über PLATO

Die PLATO AG ist ein international tätiges Softwareunternehmen mit Standorten in Lübeck, Hamburg, Dortmund und München mit dem Ziel, Unternehmen bei der Entwicklung qualitativ hochwertiger Produkte und Prozesse zu unterstützen. PLATO als Lösungsanbieter für Engineering und Compliance sorgt dafür, dass bereits in der Entwicklung mögliche Fehlerquellen identifiziert und beseitigt werden. Gerade in den frühen Stadien der Entwicklung ist eine durchgängige IT-Unterstützung hilfreich. PLATO liefert das Konzept und die Werkzeuge dazu, angefangen beim Kundenwunsch, über die Risikoanalyse (FMEA) mit darauf folgendem Manahmenmanagment, bis hin zu kompletten Dokumentationskonzepten für Entwicklung, Verbesserungs- und Projektmanagement [Brü14].

2.1.1 SCIO™

„Mit PLATO SCIO™werden kundengerechte Produkte entwickelt und produziert. PLATO SCIO™ ist eine Produktfamilie mit praxisnahen Modulen, die jeweils unterschiedliche Anforderungen und Aufgaben im Engineering erfüllen. Der besondere Erfolg von PLATO SCIO ™ liegt in der zentralen Datenbank, die alle Module gemeinsam nutzt, und das Wissensmanagement und das Wiederverwenden von Wissen erst möglich macht.“[1].

2.1.2 SCIO™-Methods

PLATO SCIO™ -Methods ist eine Web-Anwendung, die individuelle Formblätter zur Umsetzung von Engineering Methoden und Analysen für Unternehmen zur Verfügung stellt. Dabei nutzt sie alle Verwaltungs- und Stammdaten sowie Systemelemente und deren Daten, die bereits in anderen SCIO™ -Modulen angelegt wurden. Schwerpunkte von SCIO™ -Methods sind: ergänzende Informationen aufnehmen, Methoden umsetzen und Berechnungen durchführen[2].

2.1.2.1 Formblätter

Formblätter bieten dem Nutzer die Möglichkeit, relevante Daten über Funktionen, Fehlermöglichkeiten u. a. eines Produktes, methodenspezifische Daten und Werte für Kalkulationen zu erfassen und für verschiedene Zielgruppen zu präsentieren[3]. Angebotene Formblätter sind:

- FMEA

[1] http://www.plato.de/scio.html?file=tl_files/public/downloads/Produktblaetter/PLATO-SCIO%20Produktinformation%2003.pdf
[2] http://www.plato.de/scio-methods.html
[3] http://www.plato.de/scio-methods.html?file=tl_files/public/downloads/Produktblaetter/PLATO-SCIO-Methods%20Produktinformation%2002.pdf

- Funktionale Sicherheit ISO 26262/IEC 61508
- FMEDA - ISO 26262/IEC 61508
- DRBFM
- Risikoanalyse für Medizintechnik
- Gefahrenanalyse
- Anforderungsanalyse
- uvm.

2.2 Maßnahmenmanagement

Viele Menschen können sich unter dem Begriff Maßnahmenmanagement nichts vorstellen, wobei ein jeder hiermit tagtäglich konfrontiert wird. Alle Aufgaben oder Tätigkeiten, die man ausführt, sind Maßnahmen wie beispielsweise das wöchentliche Gießen der Blumen oder die täglichen Hausaufgaben, die zu erledigen sind. Das Verwalten dieser Maßnahmen nennt sich Maßnahmenmanagement.

2.2.1 Softwaregestütztes Maßnahmenmanagement

In der einfachsten Form des Maßnahmenmanagements werden die Maßnahmen gedanklich von den individuellen Personen festgehalten. Der nächste Schritt wäre es, die Maßnahmen schriftlich zu notieren, z.B. in Form eines Hausaufgabenhefts oder einer Notiz auf dem Kühlschrank. Dabei besteht die Gefahr zu vergessen, die Blumen zu gießen oder auf den Zettel mit der Aufgabe zu schauen. Hierfür eignen sich technische Hilfsmittel wie das Smartphone, das Tablet oder der Computer hervorragend. Für Privatpersonen gibt es unzählige Apps für das Smartphone oder Software für den Computer, die eingesetzt werden kann als ein Beispiel wäre remember the milk[4] zu nennen. Mit diesen Softwarelösungen können Personen Erinnerungen auf ihre Geräte erhalten, Maßnahmen sich automatisch wiederholen lassen, Maßnahmenlisten zuordnen und vieles mehr. Diese Softwarelösungen sind für Privatpersonen vom Funktionsumfang ausreichend, für Unternehmen aber unbrauchbar, da diese weit höhere Anforderungen an ein Maßnahmenmanagementsystem stellen. Ein Unternehmen möchte Maßnahmenstrukturen (siehe Abschnitt 2.2.2.2) erzeugen, Rechte verwalten können oder sicherstellen, dass Normen der entsprechenden Branchen erfüllt werden, um nur einige der Anforderungen zu nennen. In Abschnitt 2.2.2.2 wird auf diese Anforderungen näher eingegangen. Auf dem Markt gibt es einige Softwarelösungen für Maßnahmenmanagement in Unternehmen, wie zum Beispiel Babtec MM[5].

2.2.2 Allgemeines zu Maßnahmenmanagement

Folgend werden Maßnahmen und Maßnahmenstrukturen besprochen, um ein rudimentäres Verständnis für die Materie zu erhalten.

2.2.2.1 Maßnahmen

Eine Maßnahme kann eine Vielzahl an Elementen besitzen, je nach Anforderung. Typische Maßnahmen in Software für den privaten Gebrauch sind Titel, eine Beschreibung, ein Zieltermin oder der benötigte Aufwand. Unternehmen benötigen hingegen Strukturen, um die Maßnahmenprozesse zu überwachen.

[4] http://www.rememberthemilk.com/
[5] http://www.babtec.de/

Jede Maßnahme besitzt eine Person (Manager), welche für die Überwachung zuständig ist und per E-Mail über Zeitüberschreitungen oder nahende Zeitüberschreitung informiert. Des Weiteren gibt es den Verantwortlichen, der für die Umsetzung verantwortlich ist oder den Bewerter, der nach Abschluss der Maßnahme das Ergebnis bewertet, um nur zwei zu nennen. In Unternehmen ist es auch wichtig eine Kostenplanung und -überwachung durchzuführen. So werden vom Manager oder Ersteller Soll-Kosten bestimmt und der Verantwortliche trägt beim Abschließen der Maßnahme die Ist-Kosten ein. Aus diesen Daten können wertvolle Erkenntnisse gewonnen werden, wie beispielsweise eine zu großzügige Planung oder zu hohe tatsächliche Ausgaben. Mit diesen Resultaten können dann weitere Maßnahmen ergriffen werden, um diesem entgegen zu wirken.

2.2.2.2 Maßnahmenstrukturen

Die einfachste Form vom Maßnahmenmanagement sind flache Listen von Maßnahmen, welche in Unternehmen unzureichend sind. Deshalb bieten komplexere Maßnahmenmanagementsysteme Maßnahmenstrukturen. Eine Maßnahmenstruktur fängt auf oberster Ebene mit einer Maßnahme an. Dieser Maßnahme können weitere Maßnahmen zugeordnet werden, die sich dann Teilmaßnahmen nennen. Teilmaßnahmen können weitere Maßnahmen zugeordnet werden und dies lässt sich beliebig fortführen. Solche Strukturen nutzt ein jeder beim täglichen Umgang mit seinem Computer. Ordner in einem Dateisystem verhalten sich wie Maßnahmen. Auf dem Laufwerk C: lassen sich Ordner erstellen und diesen weitere Ordner zuordnen.

So würde beispielsweise in einem Unternehmen die Maßnahme erstellt werden, einen Prototyp für einen Kleinwagen zu entwickeln. Der Entwicklungsleiter der verantwortlich für diese Maßnahme ist, würde diese Aufgabe weiter aufteilen und zum Beispiel Teilmaßnahmen zu den einzelnen Komponenten erstellen wie Karosserie, Elektronik oder Motor. Der Verantwortliche der Karosserie würde darauf eine Teilmaßnahme erstellen zur Durchführung einer Designstudie oder eines 3D-Modells. Solch übergeordnete Maßnahmen können erst dann abgeschlossen werden, wenn auch alle Teilmaßnahmen abgeschlossen sind. In diesem Fall müssen Karosserie, Elektronik und Motor sowie deren Teilmaßnahmen abgeschlossen werden, bevor die Maßnahme zur Entwicklung eines Prototyps abgeschlossen werden kann. Dies ist ein sehr vereinfachtes Beispiel und würde tatsächlich viel feiner aufgeteilt werden. Der Vorteil solcher Strukturen ist, dass Aufgaben aufgeteilt werden und die Manager nur die direkten Teilmaßnahmen überwachen müssen und die Verantwortlichen dieser Maßnahmen ihre Teilmaßnahmen überwachen und so weiter.

2.3 Risikomanagement

In vielen Bereichen außerhalb des Maßnahmenmanagement fallen Maßnahmen an, die Verwaltet werden müssen, beispielshalber wird dies am Risikomanagement (PLATO SCIO^{TM}) erläutert. Mit Hilfe des Risikomanagements wird in vielen Branchen versucht, Risiken zu identifizieren, zu analysieren und ihnen entgegen zu wirken. Eine Methode, die im Risikomanagement verwendet wird, ist die Failure Mode and Effects Analysis (FMEA). Diese beschreibt, wie man vorzugehen hat und das unter anderem empfohlene Maßnahmen getroffen werden müssen, um die Risiken zu beseitigen oder zu minimieren. Ein Teil dieses Risikomanagements ist SCIO^{TM} -Methods, mit dem Formblätter (siehe Abschnitt 2.1.2.1) frei vom Kunden definiert werden können. Dies ist mit den Daten aus der Risikomanagementapplikation sowie eigenen frei wählbaren Daten möglich.

2.4 Einführung in das Semantische Web

Das „Web 2.0“ ist ein Begriff, den sicherlich ein jeder zu mindestens schon gehört hat. Hierbei handelt es sich nicht um eine Definition des World Wide Webs, sondern um einen Begriff, der den Sprung von

statischen Internetseiten („Web 1.0“) zu dynamischen Internetseiten verdeutlicht. Was genau „Web 2.0“ beinhaltet, ist nicht eindeutig, da es keine Definition ist und daher findet man auch keine einheitliche Beschreibungen hierzu. In Tabelle 2.1 sieht man eine Gegenüberstellung von „Web 1.0“ und „Web 2.0“.

Web 1.0	Web 2.0
DoubleClick	Google AdSense
Ofoto	Flickr
Akamai	BitTorrent
mp3.com	Napster
Britannica Online	Wikipedia
Persönliche Webseiten	Blogs
Spekulation mit Domain Namen	Suchmaschinen-Optimierung
Seitenaufrufe	"cost per click"
Extraktion mittels Screen Scraping	Web Services
Veröffentlichung	Beteiligung
Content Management Systeme	Wikis
Taxonomie (Verzeichnisse)	"Folksonomy" (Tagging)
Feststehend ("stickiness")	Zusammenwachsen ("syndication")

Tabelle 2.1: Unterschiede zwischen „Web 1.0“ und „Web 2.0“ [O'R09]

Diese von Tim O'Reilly erstellte Liste lässt sich beliebig erweitern und stellt nur einen Teil der Unterschiede zwischen „Web 1.0“ und „Web 2.0“ dar. Wer tiefer in das Thema „Web 2.0“ einsteigen möchte, sollte unbedingt[O'R09] lesen, worin Tim O'Reilly beschreibt, wie dieser Begriff entstanden ist und was es mit „Web 2.0“ auf sich hat.

In 2006 hat John Markoff den Begriff „Web 3.0“[6] mit dem Semantischen Web verbunden und somit als den nächsten Schritt in der Entwicklung des Internets angegeben. Das Semantische Web ist ein Standard[7] des World Wide Web Consortium (W3C). Das W3C ist eine Gemeinschaft, in der Mitgliedsorganisationen wie z.B. Microsoft oder Google, Vollzeitkräften und die Öffentlichkeit an der Entwicklung von Webstandards arbeiten, um die Entwicklung im Internet zu vereinheitlichen.

Bisher sind die im Internet dargestellten Informationen überwiegend für den Menschen aufbereitet und Maschinen können diese Daten zwar durchsuchen, aber nicht verstehen und interpretieren. So kann der Computer in einem Text nicht erkennen, ob es sich um den Namen eines Unternehmen handelt oder ob es beispielsweise ein Produktname ist. Im Semantischen Web wird deswegen auf das Resource Description Framework (RDF) zurückgegriffen. Damit eine Information für den Menschen auch verständlich bleibt, benötigt man ein Subjekt, ein Prädikat und ein Objekt. RDF bietet dies in Form von Triples an. In der Abbildung 2.1 besteht das Triple aus einem Subjekt „Rayner“, dem Prädikat „erstellt“ und dem Objekt „Maßnahme“. Durch diese Semantik wird ersichtlich, dass Rayner einer Maßnahme erstellt hat. Weitere Beispiele hierzu können in Abschnitt 4.1 gefunden werden, in dem eine geeignete Auszeichnungssprache evaluiert wird.

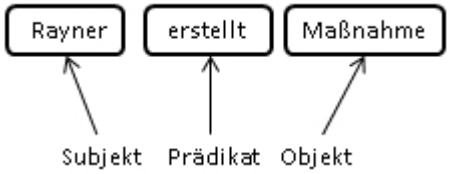

Abbildung 2.1: Beispiel eines Triples

[6] http://www.nytimes.com/2006/11/12/business/12web.html?ex=1320987600&en=254d%20697964cedc62&ei=5088
[7] http://www.w3.org/standards/semanticweb/

Verbindet man mehrere Triples miteinander, indem man die selben Subjekte oder Objekte verwendet, bilden diese einen (gerichteten) Graphen (siehe Abbildung 2.2).

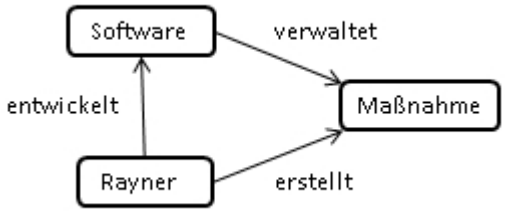

Abbildung 2.2: Beispiel eines Graphen

Subjekt, Prädikat und Objekt sind in RDF Resourcen und damit diese eindeutig in einem Graphen sind, erhalten sie einen Uniform Resource Identifier (URI). URIs sind beispielsweise in den URLs von Webseiten enthalten und stellen sicher, dass diese eindeutig sind. Für das Subjekt „Rayner“ könnte z.B. die URI http://plato.de/person/rayner verwendet werden. Das Konzept der URI lässt sich aber nicht nur auf die Triples anwenden, sondern auch auf den ganzen Graphen. Dann spricht man von named graphs. Wichtig wird dies, wenn einzelne Graphen in einem Triplestore angesprochen werden oder wenn mehrere Graphen miteinander kombiniert werden sollen.

Es ist aber nicht immer möglich, einem Subjekt, eine URI zuzuweisen. Angenommen es sollen Personen, in einem Graphen, referenziert werden, die über keine URI verfügen. In diesem Fall spricht man von „blank nodes“. Diese können im Graphen verwendet werden, da sie aber über keine eindeutig URI verfügen, können diese nicht Graphen übergreifen verwendet werden [SET09, S.66-67]. Auf Grund der Tatsache, dass in Abschnitt 3.1 eine Anforderung (R03) das Vernetzen von Graphen ist, wird in Maßnahmengraphen, von „blank nodes“ kein Gebrauch gemacht und alle Teile des Triples mit einer URI versehen.

Die letzte Ausnahme, die keine URI verwendet, ist das Literal. Darunter sind Zeichenketten zu verstehen, die Namen, Zahlen und andere Typen darstellen. Mit ihnen können Sprachen (z.B. englisch, deutsch) oder Typen (z.B. Integer, Boolean, String) verbunden werden. Üblicherweise werden Typen URIs mit dem XML Schema, einer Empfehlung des W3C, zum Beschreiben von XML Strukturen, angegeben. Z.B. werden Integer mit der URI `http://www.w3.org/2001/XMLSchema#int` definiert [SET09, S.68].

3 Anforderungsanalyse

Für einen allgemeinen Datenaustausch soll eine Ontologie entworfen (siehe Abschnitt 4.3.1) werden, die Maßnahmen aus verschiedenen Systemen einheitlich darstellt. Ziel ist es, dass das dahinter liegende System e1ns.actions über eine standardisierte Schnittstelle auf der Basis des Semantischen Web mit allen Maßnahmen identisch umgehen kann, ohne die Schnittstelle bei jedem weiteren System anpassen zu müssen. In Abschnitt 3.1 werden die in der Einleitung teilweise erwähnten Anforderungen zusammengefasst, die an das Maßnahmenmanagementsystem e1ns.actions gestellte werden.

Diesbezüglich werden in diesem Kapitel unter anderem bestehende Maßnahmen- und Projektmanagementsysteme auf dem Markt analysiert, geprüft ob es bereits Lösungen für dieses Problem gibt und ob diese den zuvor ermittelten Anforderungen genügen.

3.1 Allgemeine Anforderungen

R01 Bei der PLATO AG wird für das serverseitige Programmieren, ausschließlich die Programmiersprache Python in der Version 2.7 verwendet. Aus Gründen der Wartbarkeit und der Kompatibilität von Standardmodulen zu anderen Produkten der PLATO AG, wie z.B. SCIO^{TM}-Methods, muss die Implementierung mit Python 2.7 erfolgen.

R02 Die Ontologie benötigt ein Standardmodell für Maßnahmen

R03 Das Maßnahmenmanagementsystem soll in der Lage sein, Graphen als named graphs darzustellen. Wie in Abschnitt 2.4 beschrieben werden named graphs benötigt, um einzelne Graphen ansprechen zu können und wie später in Kapitel 4 beschrieben, ist dies von Nöten, um unterschiedliche Graphen miteinander zu vernetzen.

R04 Auszugebende RDF Dateien müssen vom Menschen möglichst einfach zu lesen sein.

R05 Ein auszuwählender Triplestore (siehe Abschnitt 4.2) muss die ausgewählte Syntax (siehe Abschnitt 4.1.7) und weitere Notationen beherrschen, um ein breites Spektrum abzudecken, sodass die RDF Dateien aus den verschiedenen externe Systeme ausgelesen werden können.

R06 Eine umfangreiche Dokumentation für den Triplestore.

R07 Eine große und lebendige Community, um das Arbeiten mit dem Triplestore so einfach wie möglich zu gestalten und bei Fragen mögliche Ansprechpartner zu haben.

R08 Die zu erwartenden Datenmengen sind nicht abzuschätzen, jedoch wäre es wünschenswert, wenn der Triplestore mit großen Datenmengen umgehen kann, um auch für die Zukunft gerüstet zu sein.

R09 Der Triplestore soll Open Source sein und somit die Möglichkeit bieten, eigene Änderungen vorzunehmen.

R10 Bei der Lizensierung des Triplestores soll es sich nicht um eine Copyleft-Lizenz[1], wie beispielsweise GPL[2], halten. Es geht hierbei nicht darum, der Open Scource Community Programmcode vorzuenthalten, sondern hält die Option möglichst flexible vorzugehen offen.

[1] Bei Veränderungen muss es unter der selben Lizenz weitergegeben werden.
[2] General Public License

Im Zuge von Kundenprojekten kann es vorkommen, dass diese Klauseln vereinbaren, die es der PLATO AG verbieten Programmcode/Funktionalität an Mitbewerber weiterzugeben. Durch eine Copyleft-Lizenz wäre dies nicht möglich. Die PLATO AG ist jedoch stets bemüht, Open Source Projekten, von denen sie profitiert, etwas zurückzugeben und zu dem Projekt beizutragen.

R11 Daten im Triplestore müssen mit SPARQL abfragbar sein.

3.2 Ermittlung der Maßnahmenattribute für ein Standardmodell

Für die Entwicklung eines semantischen Modells müssen die hier untersuchten Systeme dahingehend analysiert werden, was allen als Kern gleich ist und für ein Standardmodell dienen könnte und was Variationen sind und wie Erweiterungen am einfachsten eingebunden werden können.

Die folgenden Systeme tauchen am häufigsten bei Kunden der PLATO AG[3] auf und müssen über Schnittstellen oder durch Ex- und Import von Dateien Daten austauschen. Um auch den Markt der individuellen Personen abzudecken, wurde die Liste exemplarisch um einen Offline-Client und einer Webanwendung erweitert, da es unzählige Tools auf den Markt gibt und eine qualifizierte Auswahl den Zeitrahmen dieser Ausarbeitung gesprengt hätte.

3.2.1 SAP-Meldungen

Für Maßnahmen in SAP[4] ist es sehr schwer, an Informationen zu kommen, da sich SAP hier sehr bedeckt hält. Folgende Attribute sind laut Onlinedokumentation[5] verfügbar.

- Schlüssel
- Arbeitsanweisung
- Geplanter Beginn
- Geplantes Ende
- Status

Der Abbildung 12.6 in [HS05, S. 483] konnten noch folgende Attribute entnommen werden.

- Bezeichnung (Arbeitsanweisung)
- Planstart (Geplanter Beginn)
- Zu erledigen bis (Geplantes Ende)
- Iststart
- Istende
- Verantwortlicher

SAP bietet drei Programmierschnittstellen an, die RFC Libary, das Business Application Programming Interface (BAPI) und den Java Connector (JCo) [MN04, S. 11]. Im Umfeld der SAP-Software steht der Begriff RFC für Remote Function Call und ermöglicht den Aufruf von Funktionen von externen Systemen. Die SAP Schnittstellen sind ein sehr komplexes Thema und werden im dieser Arbeit nicht weiter erläutert. Für weiterführende Literatur ist [MN04] zu empfehlen, in dem auf alle drei Programmierschnittstellen und ihre Vor-, sowie Nachteile eingegangen wird.

[3] http://www.plato.de/
[4] http://www.sap.com/germany/index.html
[5] http://help.sap.com/saphelp_46c/helpdata/de/c4/543d3854126956e10000009b38f842/frameset.htm

3.2.2 Babtec.MM

Die Babtec Informationssysteme GmbH bietet mit Babtec.Q eine CAQ [6] Software für ganzheitliche Produkt- und Prozessoptimierung. Eines der Module ist Babtec.MM, welches sich ausschließlich mit Maßnahmen befasst. Abbildung 3.1[7] zeigt eine solche Maßnahme.

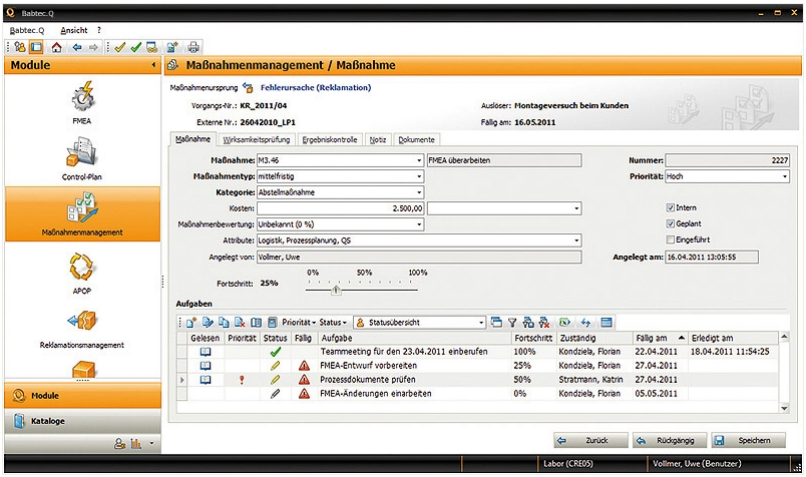

Abbildung 3.1: Dialog zum Bearbeiten einer Maßnahme in Babtec.MM

Eine Maßnahme in dieser Anwendung besteht aus folgenden Attributen.

- Maßnahme (Titel)
- Maßnahmentyp
- Kategorie
- Kosten
- Maßnahmenbewertung
- Attribute
- Angelegt von
- Angelegt am
- Nummer
- Priorität
- Fortschritt
- Checkboxen (weitere Typ Spezifizierung)
- Teilmaßnahmen
- Fällig am
- Auslöser (der Maßnahme)
- Vorgangs-Nr.
- Externe Nr.
- Maßnahmeursprung

[6] Computer-aided quality assurance - rechnerunterstützte Qualitätssicherung
[7] http://www.babtec.de/uploads/pics/il_produkte_mm_r6_2013.jpg

- Notizen
- Status
- Zuständig

Bezüglich Schnittstellen wird ein Tool (Babtec.SI/APICONNECT) speziell für die Integration zu SAP angeboten sowie ein Integrationtool (Babtec.SI/ERP) für PPS/ERP-Systemen wie z.B. für Infor ERP, proALPHA und Microsoft Dynamics. Anbindungen an unternehmensspezifische Eigenentwicklungen müssten im Zuge eines Projektes erstellt werden.

3.2.3 Qualitainer MM

Qualitainer CAQ-Software bietet wie Babtec eine CAQ Software als modulare Komplettlösung an, die das Modul Qualitainer MM enthält. Die folgenden Attribute einer Maßnahme von diesem Tool wurden dem Produktvideo[8] entnommen, da die Firma hierzu ansonsten keine Informationen bereit stellt.

- Typ
- Titel
- Beschreibung
- Auftraggeber
- Autragnehmer
- Priorität
- Planbeginn
- Planende
- Ist Beginn
- Ist Ende
- Wirksamkeit
- Erledigungsgrad
- Status
- Teilmaßnahmen (TODOs)

Eine offene Standardschnittstelle wird von Qualitainer bereitgestellt. Diese bietet die Möglichkeit, übergeordnete Systeme anzubinden, und unterstützt Stamm- sowie Bewegungsdaten.

3.2.4 ToDoList

ToDoList ist ein Open-Source Projekt[9] von Daniel Godson. In der Abbildung 3.2[10] sieht man die geöffnete Applikation.

[8] http://www.qualitainer.de/Massnahmenmanagement.mp4
[9] http://www.abstractspoon.com/tdl_resources.html
[10] http://www.abstractspoon.com/tdl_resources_files/todolist.png

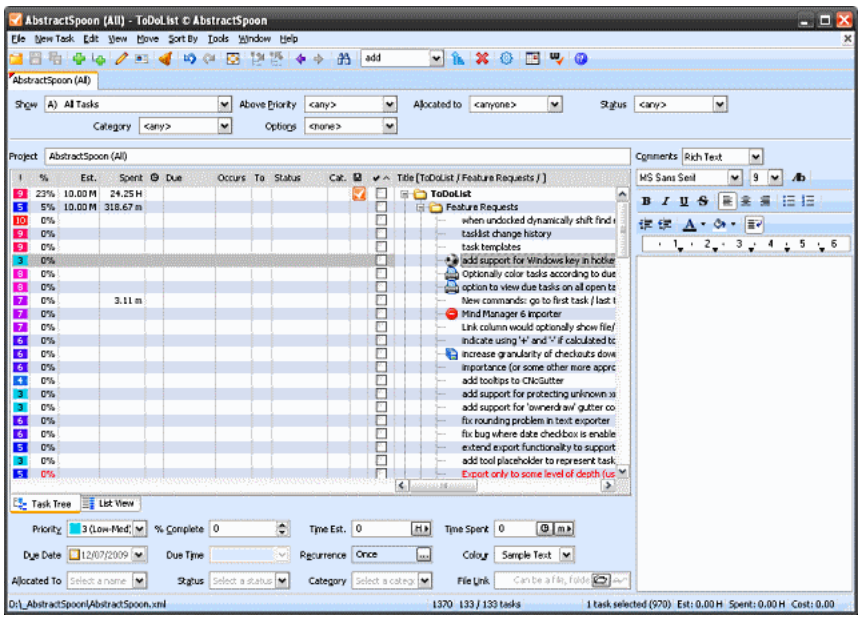

Abbildung 3.2: Maßnahmenliste und Maßnahme aus ToDoList

Die Attribute der Maßnahme in ToDoList lauten wie folgt:

- Title
- Priorität
- Abgeschlossen in Prozent
- Geschätzter Aufwand
- Tatsächlicher Aufwand
- Zieltermin
- Zielzeit
- Wiederholung der Maßnahme
- Zugeordnet zu (Person)
- Status
- Kategorie
- Beschreibung

Diese Anwendung verfügt über keine Schnittstelle, sondern speichert die Daten lediglich als XML.

3.2.5 Remember The Milk

Bei Remember The Milk[11] handelt es sich um ein einfaches, aber dennoch sehr populäres Maßnahmenmanagmenttool. Im Google Playstore [12] wurde es bereits über 1 Millionen mal heruntergeladen und hat bei fast 27.000 Bewertungen knapp über 4 von 5 Sternen. Neben einer Android App gibt es Apps für Apple und BlackBerry Geräte. Desweiteren bietet es Integrationsmöglichkeiten für Gmail, Microsoft Outlook, Evernote oder Twitter an. Natürlich verfügt es auch über eine Weboberfläche, mit der man seine Maßnahmen (hier Aufgaben) verwalten kann.

In Abbildung 3.3 sieht man die sehr einfach gehaltene Maßnahme, die folgende Attribute beinhaltet.

[11] https://www.rememberthemilk.com/
[12] https://play.google.com/store/apps/details?id=com.rememberthemilk.MobileRTM

Abbildung 3.3: Aufgabe in Remember The Milk

- Title
- Liste
- Fällig
- Wiederholen
- Zeitschätzung
- Tags
- Ort
- URL
- Aufgeschoben
- Freigegeben für
- Notizen

3.2.6 Schlussfolgerung

Keine der untersuchten Anwendungen, bietet die Möglichkeit, die Daten maschinenlesbar auszuwerten, was wiederum bedeutet, dass dieser Markt noch unerschlossen ist und eine hervorragende Möglichkeit bietet, sich hiermit ein Alleinstellungsmerkmal zu schaffen. Um in Abschnitt 4.3.1 eine umfassende Ontologie entwerfen zu können, wurden die verwendeten Attribute gegenüber gestellt und allgemeingültige Attribute herausgearbeitet.

SAP	Babtec.MM	Qualitainer MM	ToDoList	Remember The Milk	Abgeleiteten Attribute
-	Maßnahme (Title)	Titel	Titel	Titel	Titel
Schlüssel	Vorgangs-Nr. / Externe Nr.	-	-	-	Schlüssel
Arbeitsanweisung	-	Beschreibung	Beschreibung	-	Arbeitsanweisung
-	Notizen	-	-	Notizen	Umsetzungsbesch.
Geplanter Beginn	-	Planbeginn	-	-	
Iststart	-	Ist Beginn	-	-	Starttermin
Geplantes Ende	-	Planende	-	-	-
Istende	Fällig am	Ist Ende	Zieltermin / Zielzeit	Fällig	Zieltermin
Status	Status	Status	Status	-	Status
-	Fortschritt	Erledigungsgrad	Abgeschlossen in Prozent	-	Umsetzungsgrad
-	Priorität	Priorität	Priorität	-	Priorität
Verantwortlicher	Zuständig	Autragnehmer	Zugeordnet zu (Person)	-	Verantwortlicher
-	-	Auftraggeber	-	-	Manager
-	Angelegt von	-	-	-	Ersteller
-	Angelegt am	-	-	-	Erstelldatum
-	Kategorie	-	Kategorie	Liste	Kategorie
-	Maßnahmentyp	Typ	-	-	-
-	Checkboxen (Typ Spezifizierung)	-	-	Tags	-
-	Teilmaßnahmen	TODOs	-	-	-
-	-	-	Geschätzter Aufwand	Zeitschätzung	-
-	-	-	Wiederholung der Maßnahme	Wiederholen	-
-	Kosten	-	-	-	-
-	Maßnahmebewertung	-	-	-	-
-	Attribute	-	-	-	-
-	Nummer	-	-	-	-
-	Auslöser (der Maßnahme)	-	-	-	-
-	Maßnahmeursprung	-	-	-	-
-	-	Wirksamkeit	-	-	-
-	-	-	Tatsächlicher Aufwand	-	-
-	-	-	-	Ort	-
-	-	-	-	URL	-
-	-	-	-	Aufgeschoben	-
-	-	-	-	Freigegeben für	-

Tabelle 3.1: Gegenüberstellung aller Attribute

Folgende Attribute sind in den meisten Systemen wiederzufinden und daher als allgemeingültig anzusehen. Einige der ausgewählten Attribute traten nur in einzelnen Anwendungen auf, haben es aber dennoch in die Auswahl geschafft. Hierzu gehört zum einen der Schlüssel. Dieser wurde nur in zwei Systemen erwähnt, wird aber sicherlich auch in den anderen Anwendungen vorhanden sein, da jede Maßnahme in irgendeiner Weise identifiziert werden muss. Anzunehmen ist, dass dem Benutzer in den anderen Tools diese Information lediglich nicht angezeigt wird. Die Systeme mit Notizen hatten im Gegensatz zu den anderen keine Beschreibung. Dies übernimmt in diesen Anwendungen der Titel der Maßnahme. Für das allgemeingültige Maßnahmenmodell wird deshalb Arbeitsanweisung und Umsetzungsbeschreibung genommen. Bei den Termin gab es Tools die die Planung sehr genau nehmen und andere die nur feste Termine vergeben. Einfachheitshalber wird in diesem Modell nur Starttermin und Zieltermin genommen und später ggf. auf Plantermine erweitert. Wie bei dem Schlüssel wird der Ersteller und das Erstelldatum in jedem System vorhanden sein, jedoch nicht immer angezeigt werden. Des Weiteren wird in den meisten Systemen der Ersteller auch als Auftraggeber gesehen, jedoch muss dies nicht unbedingt der Fall sein. Um hier flexibler zu sein, wird dies im allgemeingültigen Modell in Ersteller und Manager getrennt und kann wenn gewollt ein und dieselbe Person sein.

Attribute	Beschreibung
Titel (Name)	Kurze Beschreibung der Maßnahme
Schlüssel (Key)	Hiermit kann die Maßnahme eindeutig identifiziert werden
Arbeitsanweisung (Definition)	Beschreibung der Aufgabe
Umsetzungsbeschreibung (Implementation)	Dokumentation der Umsetzung der Maßnahme
Starttermin (Start date)	Ab wann mit dem Umsetzen der Maßnahme begonnen werden kann
Zieltermin (End date)	Termin, an dem die Maßnahme abgeschlossen sein soll
Status (Status)	Gibt an, in welchem Arbeitsschritt sich die Maßnahme befindet (Bsp.: “Entwurf”)
Umsetzungsgrad (Implementation date)	Gibt an, wie weit die Umsetzung vorangeschritten ist (Bsp.: 25%)
Priorität (Priority)	Wichtigkeit einer Maßnahme
Verantwortlicher (Owner)	Auftragnehmer
Manager (Manager)	Auftraggeber
Ersteller (Creator)	Person, die die Maßnahme angelegt hat
Erstelldatum (Creation date)	Datum der Erstellung
Kategorie (Category)	Gibt an, welcher Kategorie die Maßnahme zugeordnet wird

Tabelle 3.2: Attribute einer allgemeingültigen Maßnahme

4 Architektur

In Abbildung 4.1 sieht man die Grobarchitektur, die in Teilen, der Einleitung schon beschrieben wurde. Wie in Abschnitt 2.3 erwähnt, fallen auch in anderen Produkten von PLATO, wie z.B. SCIO™(siehe Abschnitt 2.1.1) Maßnahmen an. Hinzu kommen Projekte, an denen die Maßnahmen in e1ns.actions hängen, die in SCIO abgelegt sind. Um an diese Daten zu gelangen, stellt SCIO™ eine Programmierschnittstelle namens SAPI[1] zur Verfügung. All diese unterschiedlichen Maßnahmen, denen zur Zeit verschiedene Modelle zu Grunde liegen, werden mit Hilfe einer Ontologie so aufbereitet, dass diese über den Triplestore Provider durch externe Systeme abfragbar sind, ohne ein tieferes Verständnis des dahinterliegenden Modells zu haben. Dieser Triplestore Provider wird über SPARQL-Queries durch externe Anwendungen ansprechbar sein. E1ns.actions wird gleichzeitig ein Endpoint für alle Produkte von PLATO sein, sodass hier die Ontologie für Maßnahmen mit anderen Ontologien verbunden werden können. Neben den SPARQL Anfragen wird es ein Ex- und Import Modul geben, welches RDF Dateien erzeugt und auslesen kann. Hierüber können Systeme kommunizieren die über keine SPARQL Anbindung verfügen.

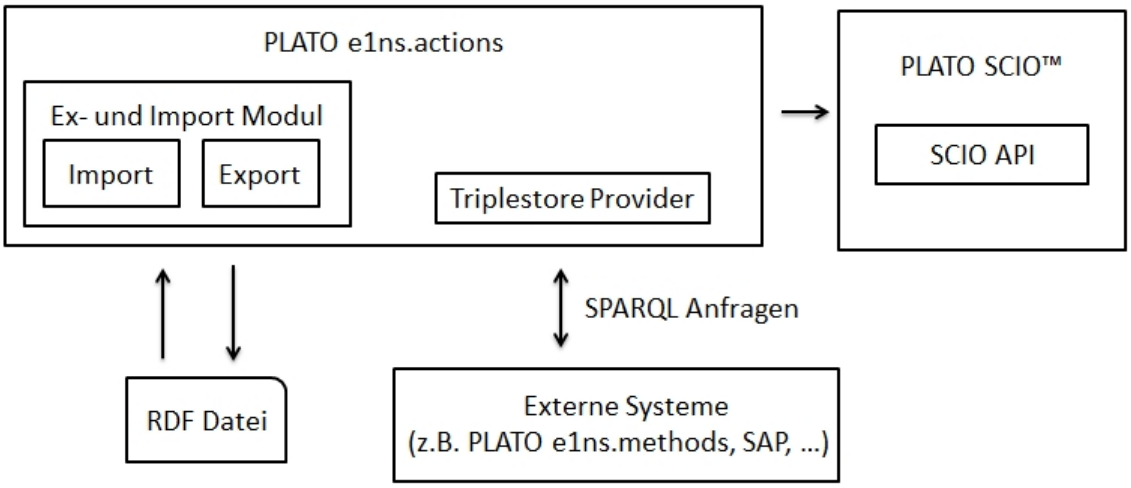

Abbildung 4.1: Grobarchitektur

4.1 Auswahl einer RDF-basierten Notation

In diesem Kapitel wird nach einer RDF-basierten Notation gesucht, die die Triples am besten beschreibt, gut zu lesen ist und named graphs darstellen kann.

4.1.1 N-Triples

N-Triples sind eine zeilenbasierte, einfache, aber wortreiche Notation. Jede ausgegebene Zeile stellt eine Aussage dar, die sich aus Subjekt, Prädikat und Objekt zusammensetzen. Abgeschlossen wird eine

[1] SCIO application programming interface

Aussage mit einem Punkt. In Listing 4.3 sieht man ein solches Triple sowie auch ein Beispiel, von einem „blank node“ `_:Subjekt` und ein Literal „`Objekt`“.

```
1 <http://beispiel.uri/resources/Subjekt> <http://beispiel.uri/resources/Praedikat> <http
      ://beispiel.uri/resources/objekt> .
2 _:Subjekt <http://beispiel.uri/resources/Praedikat> "Objekt" .
```

Listing 4.1: Beispiel für N-Triples

Diese Einfachheit ist speziell bei manuell erstellten Datensätzen für das Testen und die Fehlersuche in Anwendungen geeignet [SET09, S. 70-71]. N-Triples stammen aus dem komplexeren N3 (siehe Kapitel 4.1.2) und stellen eine Untermenge dessen dar [HKRS08, S. 40]. Auf den ersten Blick sind diese kaum zu unterscheiden und daher bekommen Dateien mit N-Triples die Dateiendung “.nt”. Auch Turtel (siehe Kapitel 4.1.3) ist eine Übermenge von N-Triples, daher können diese auch in allen Tools geöffnet werden, die N3 und Turtel unterstützen. Wie N3 und Turtel sind N-Triples nicht auf XML angewiesen und dadurch leichter lesbar sowie manuell einfacher zu bearbeiten als z.B. RDF/XML (siehe Kapitel 4.1.5).

4.1.2 Notaion 3 (N3)

Die beschriebene Einfachheit von N-Triples hat auch seine Nachteile. Wie in Listing 4.3 zu sehen, gibt es einige Wiederholungen [SET09, S. 72]. Bei kleinen Datenmengen macht dies keinen Unterschied, arbeitet man aber mit großen Mengen an Daten, kostet es zusätzlich Zeit, diese zu übermitteln und auszuwerten. Deshalb hat N3 Präfixe, mit denen sich URIs durch URI Präfixe abkürzen lassen. Nehmen wir eine URI aus Listing 4.3 und weisen dieser einem Präfix zu.

```
1 @prefix res: <http://beispiel.uri/resources/> .
```

Listing 4.2: N3 Prefix

Jetzt kann anstatt der langen URI der Präfix verwendet werden.

```
1 res:Subjekt res:Praedikat res:Objekt .
```

Listing 4.3: Beispiel für N-Triples

Für N3 wird die Dateiendung “.n3” verwendet, um wie in Kapitel 4.1.1 beschrieben, diese von N-Triple und Turtle Dateien einfach unterscheiden zu können.

4.1.3 Turtle

Turtel stellt eine Untermenge von N3 und eine Übermenge von N-Triples dar, überschreitet anders als N3 das Graphenmodell von RDF aber nicht. N3 kann „[...] auch komplexere Ausdrücke wie Pfade und Regeln beinhalten.“ [HKRS08, S. 40]. Turtle nutzt so die Einfachheit von N-Triples, um Graphen darzustellen und praktische Kurzschreibweisen aus N3 [HKRS08, S. 40]. Aufgrund der bereits erwähnten Ähnlichkeit haben Dateien, die mit Turtel geschrieben sind die Endung “.ttl”.

4.1.4 TriG

TriG erweitert Turtel um Triples in multiplen Graphen zu gruppieren[2], ist ansonsten jedoch nahezu identisch.

[2] http://www.w3.org/TR/trig/

```
@prefix res: <http://beispiel.uri/resources/> .
@prefix graph: <http://beispiel.uri/graph/>
graph:G01 { res:Subjekt res:Praedikat res:Objekt .
     res:Subjekt01 res:Praedikat01 res:Objekt01 . }
graph:G02 { graph:G01 res:Praedikat02 res:Objekt03 . }
```

Listing 4.4: Beispiel für drei verknüpfte named graphs

4.1.5 RDF/XML

RDF/XML ist eine weitere Beschreibungssprache für RDF und wird häufig fälschlicherweise für das gleiche gehalten. Dies rührt daher, da das W3C bei der ursprünglichen Empfehlung für RDF gleichzeitig die Beschreibung von RDF als Datenmodell und XML als Beschreibung des RDF Modells gearbeitet hat. Man muss unterscheiden das RDF/XML die erste, aber nicht die einzige Beschreibungssprache für RDF ist, wie es die vorangegangen und nachfolgenden Kapitel zeigen[SET09, S.73]. In Listing 4.5 sieht man ein Beispiel, vom W3C[3], zu RDF/XML.

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
    </rdf:Description>
  </ex:editor>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:fullName>Dave Beckett</ex:fullName>
    </rdf:Description>
  </ex:editor>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <dc:title>RDF 1.1 XML Syntax</dc:title>
</rdf:Description>
```

Listing 4.5: Beispiel für RDF/XML

Wie in Kapitel 4.1.3 bereits angemerkt, werden andere Notationen dieser vorgezogen, da diese Struktur für Menschen schwerer zu erkennen und bearbeiten ist.

4.1.6 RDFa

RDFa ist keine reine RDF Beschreibungsnotation, sondern ermöglicht es, (X)HTML Seiten um maschinenlesbare RDF Daten zu ergänzen. Der Vorteil hierin besteht, dass bereits bestehende (X)HTML Seiten, die für den Menschen gedacht sind, einfach mit Information für Maschinen ergänzt werden, sodass nichts doppelt gepflegt werden muss und es für Maschinen sowie für Menschen lesbar ist.

In Tabelle 4.1 [SET09, S. 77-78] werden alle Attribute von RDFa beschrieben.

[3] http://www.w3.org/TR/rdf-syntax-grammar/

Attribut	Beschreibung
about	Eine URI die zur Festlegung des Subjekts benötigt wird
rel	Beschreibt die Beziehung zwischen zwei Resourcen
property	Beschreibt die Beziehung zwischen einer Resource und einem Literal
rev	Beschreibt die Beziehung zwischen zwei Resourcen in umgekehrter Reihenfolge
content	Ein String der ein Literal darstellt
href	URI Resource die ein Objekt beschreibt (Hyperlink)
src	URI Resource die ein Objekt beschreibt (Quelle Bilddatei)
resource	Beschreibt Subjekt oder Objekt
datatype	Datentyp vom Literal
typeof	Typ des Subjekts

Tabelle 4.1: Attribute von RDFa

Im Beispiel[4] aus Listing 4.6 wird ein Staubsauger mit RDFa beschrieben, damit Suchmaschinen diese Informationen auslesen können.

```
<div vocab="http://schema.org/" typeof="Product">
  <p>Kaufen Sie den
    <span property="name">Staubsauger XF704</span>
    jetzt im Sonderangebot!
    <img property="image" src="acmeXF704.jpg" />
  </p>
  <p prefix="dc: http://purl.org/dc/elements/1.1/" resource="acmeXF704.jpg">
    (Produktabbildung: Foto'
    <span property="dc:title">Sauberkeit</span>
    ' von
    <span property="dc:creator">Max Mustermann</span>
    ,
    <span property="dc:rights">
      freigegeben zur weiteren Verwendung ohne Einschränkung)
    </span>
  </p>
</div>
```

Listing 4.6: Beispiel für RDFa

4.1.7 Fazit

Wie man in den vorausgegangenen Abschnitten sehen konnte, gibt es zwei Gruppen von Notationen, die XML basierten und die nicht XML basierten Notationen. Beschreibungssprachen wie RDF/XML und RDFa haben ihre Vorteile, beispielsweise, wie bereits in Kapitel 4.1.6 beschrieben, können bestehende (X)HTML Seiten um RDFa erweitert werden, um sie für Maschinen lesbar zu machen und man so nicht zwei unterschiedliche Dateien pflegen muss. Nichtsdestotrotz sind XML basierte Notationen schwerer für Menschen zu lesen und von daher wird die Wahl auf eine nicht XML basierte Beschreibungssprache fallen.

N-Triples sind durch die ständigen Wiederholungen unpraktisch und blähen den Dateninhalt unnötig auf. Aufgrund der großen Ähnlichkeit von Turtle und TriG macht es kaum einen Unterschied, welche Syntax man nimmt. Durch die Möglichkeit, Triples in multiplen Graphen zu gruppieren, macht es jedoch Sinn, TriG über Turtle zu nehmen. Dies lässt nur noch TriG und N3 über. N3 ist zwar umfangreicher als TriG, jedoch hat TriG, da es seine Wurzeln in Turtel hat, eine sehr ähnlich Syntax zu SPARQL. Dies

[4] http://de.wikipedia.org/wiki/RDFa

sollte das spätere Implementieren erleichtern, da man sich auf eine und nicht wie im Fall von N3 auf zwei Syntax konzentrieren muss.

4.2 Auswahl eines geeigneten Triplestores

Ein Triplestore ist eine auf Triples optimierte Datenbank und ähnelt einer relationale Datenbank sehr. Man kann mit ihnen Daten speichern und Daten über eine Anfragesprache wie bei einem Triplestore über SPARQL abfragen. Es wäre möglich über eine relationale Datenbank Triples zu speichern und abzufragen. Der Unterschied zu einem Triplestore ist wie bereits erwähnt, dass diese auf Triples optimiert ist und mit Millionen von Triples arbeiten kann, was in der Praxis nicht ungewöhnlich ist.

In diesem Kapitel wird nach einem geeigneten Triplestores für dieses Projekt gesucht. Die Anforderung R10, schloss Triplestores wie z.B. Virtuoso, Oracle, 4store oder Bigdata aus. Mit dieser und den Anforderungen, die bereits in Abschnitt 3.1 angesprochen wurden, fiel die Auswahl auf die in den folgenden Abschnitten vorgestellten Triplestores.

4.2.1 Sesame

Sesame is ein Open Source Java Framework mit einer BSD-style license. Im Zuge eines Forschungsprojektes wurde es von der Firma Aduna entwickelt und verfügt über eine aktive Community, die sich an der Entwicklung beteiligt.

Sesame stellt eine ausführliche Dokumentation[5] zur Verfügung und eine leicht zu benutzende API. Es ist sehr flexible und bietet die Möglichkeit, die Daten in relationalen Datenbanken, Dateisystem, keyword indexer, usw. zu speichern. Die SPARQL Abfragesprache wird voll unterstützt und unterstützt alle gängigen RDF Dateiformate wie z.B. RDF/XML, Turtle, N-Triples und TriG.

> (10-20 million triples) " is a lot, but most serious triple stores can handle this I'd say. Sesame certainly can, ..."
> (Jeen Broekstra)

10 bis 20 Millionen Triples sollen das Minimum sein, was Sesame behandeln kann. In einem durchgeführten Benchmark[6] wurden ungefähr 70 Millionen RDF Triples behandelt, welches aber nicht das Maximum darstellt, sondern Sesame soll mit deutlich mehr umgehen können, speziell wenn die Hardware erweitert wird. Hierbei handelt es sich jedoch nicht um Fakten, sondern um Vermutungen.

4.2.2 Mulgara

Wie Sesame ist Mulgara in Java geschrieben und steht unter der Apache 2.0 Lizenz. Bei Mulgara handelt es sich um einen Fork vom Kowari Projekt. Neben SPARQL wird auch TQL als Abfragesprache unterstützt.

Es werden keine relationalen Datenbanken unterstützt, sondern eine für das Speichern und Verwalten von Metadaten optimierte Datenbank.

> "The Mulgara triple store is scalable up to 0.5 billion triples (with 64-bit Java)"
> (Norman Gray)

Mulgara soll skalierbar bis zu 0,5 Milliarden Triples sein, jedoch gibt es hierzu keine Benchmarks. Eine Dokumentation ist vorhanden, aber sehr lückenhaft. Die Entwickler[7] selbst empfehlen unter anderem, sich den Programmcode herunter zu laden, sollte man Fragen zur Architektur haben. Eine Community ist nicht vorhanden und der letzte Release erfolgte 2012.

[5] http://www.openrdf.org/documentation.jsp
[6] http://www.openrdf.org/forum/mvnforum/viewthread?thread=829
[7] http://code.mulgara.org/projects/mulgara/wiki/FAQ

4.2.3 Jena

Jena wurde ursprünglich von Forschern der HP Labs entwickelt. 2009 entschied sich HP, die direkte Entwicklung einzustellen, jedoch das Projekt weiterhin zu unterstüzen. Im darauffolgenden Jahr wurde Jena in die Apache Software Foundation aufgenommen und 2012 ein Top-Level Project. Jena war von Anfang an ein Open-Source Projekt und steht unter der Apache 2.0 Lizenz.

Bei Jena handelt es sich um ein Java Framework, welches auch über HTTP (Fuseki) angesprochen werden kann. Es bietet Unterstützung für die Formate RDF/XML, Turtle, N-Triples, sowie RDFa. Neben diesem stellt es eine Ontologie API zur Verfügung, die OWL und weiteren Ontologie Sprachen unterstützt. Wie die anderen Triplestores kann die Abfragesprache SPARQL verwendet werden. Als Speicher bietet Jena eine Reihe an Möglichkeiten. So können Arbeitsspeicher, SDB (SQL Datenbanken), TDB (native tuple stores) oder aber auch beliebig angepasste Speicher verwendet werden.

Jena Verfügt über eine hervorragende Dokumentation[8] und hat eine ganze Reihe an Tutorials[9], um den Einstieg zu erleichtern. An Jena wird aktive Community[10], die die Entwicklung vorantreibt.

Leigh Dodds sagt Folgendes zu Jena unter Postgres.

> "Our store is pretty big – its about 200M triples. We're currently using Jena on Postgres. For our needs this worked out better than Jena/MySQL, Sesame, and Kowari."
> (Leigh Dodds)

Jena TDB[11] konnte 1,7 Milliarden, davon 1,5 Milliarden einmalige Triples innerhalb von 36 Stunden laden, was einem Durchsatz von 12.000 triples/s entspricht.

4.2.4 RDFLib

RDFlib ist abweichend zu den bereits vorgestellten Triplestores nicht mit JAVA umgesetzt, sondern eine reine Python Implementierung. Es steht unter der BSD Lizenz, verfügt über eine aktive Community [12] und eine gute Dokumentation[13].

Es unterstützt eine Reihe von Formaten wie z.B. RDF/XML, N3, Turtle und RDFa und kann wie alle anderen Triplestores mit SPARQL umgehen. RDFLib bietet die Verwendung des Arbeitsspeichers sowie das persistente Speichern auf einer Berkeley DB an. Über eine Plugin-Architektur lassen sich auch andere Datenbanken anbinden. Beispielsweise gibt es bereits Plugins für sqlite, postgres oder mysql.

4.2.5 Fazit

Bei der Analyse der einzelnen Triplesstores stellte sich schnell heraus, dass Mulgara es nicht in die engere Auswahl schaffen würde. Bei der Wahl eines Tools, egal für welches Einsatzgebiet, ist es entscheidend, dass es aktiv weiterentwickelt wird und über eine ausreichend große Community verfügt die dieses Tool unterstützt. Beides ist bei Mulgara nicht gegeben. Der letzte Nachrichteneintrag ist zwar vom 10.Februar.2014, jedoch nur mit dem Versprechen, dass die Entwickler noch da sind und das Projekt noch nicht gestorben ist. An den Bug-Tickets erkennt man auch, dass es keine nennenswerte Community mehr gibt. Der letzte Eintrag ist vom 07.02.2012.

[8] http://jena.apache.org/documentation/javadoc/jena/
[9] http://jena.apache.org/tutorials/index.html
[10] https://issues.apache.org/jira/browse/JENA/?selectedTab=com.atlassian.jira.jira-projects-plugin:summary-panel
[11] http://seaborne.blogspot.com/2008/06/tdb-loading-uniprot.html
[12] https://github.com/RDFLib/rdflib/graphs
[13] http://rdflib.readthedocs.org/en/latest/

#	Tracker	Status	Priorität	Thema	Zugewiesen an	Aktualisiert
213	Bug	New	Normal	Error on ORDER BY variables projected away by SELECT clause		07.02.2012 21:37
210	Bug	New	High	org.mulgara.sparql.parser.cst.AndExpression contains wrong assertion	Paul Gearon	05.01.2010 06:40
208	Bug	New	High	Subqueries not supported via REST	Paul Gearon	20.10.2009 08:42
207	Bug	New	High	org.mulgara.protocol.StreamedSparqlJSONObject.emit() produces improper JSON	Paul Gearon	14.10.2009 07:18
206	Bug	New	Normal	MulgaraUserConfig uses system classloader for conf/mulgara-x-config.xml	Paul Gearon	27.09.2009 18:54
204	Bug	New	High	TQL Causing Mulgara to hang (regression from 2.0.9 to 2.1.3)	Paul Gearon	16.09.2009 23:33

Abbildung 4.2: Mulgara Tickets

Sesame und RDFLib scheinen zwei sehr gute Triplestores zu sein. RDFLib durch die Implementierung in Python wäre in der späteren Maßnahmenapplikation besser anzusprechen, da diese auch in Python programmiert wird (siehe Kapitel 5). Sesame hat mit bis zu 100 Millionen Trilpes eine sehr gute Performanz, lässt danach aber nach und kann große Datenmengen über dieser Grenze nicht mehr so schnell laden[14]. Durch den Ursprung in der Forschung verfügt Jena über eine Vielzahl an neuen und experimentellen Bibliotheken, hat jedoch auch den Nachteil, dass es durch diesen Umfang eine sehr steile Lernkurve hat. Auch das Ansprechen über HTTP (Fuseki) macht es leicht, mit Jena zu kommunizieren und benötigt keine Schnittstelle zu JAVA. Letztendlich wurde sich für Jena entschieden, da dieses durch den großen Umfang und die Flexibilität in allen Bereichen glänzen kann. Am wichtigsten jedoch waren die sehr gute Dokumentation sowie die angebotenen Tutorials. Desweiteren eine aktive und große Community, die eine Implementierung deutlich erleichtert und sicherstellt, dass bei auftretenden Problemen Ansprechpartner zur Verfügung stehen und das Ganze somit relativ zukunftssicher ist. Durch diese Argumente kann auch die steile Lernkurve in Kauf genommen werden.

4.3 Ontologie

Der Begriff „Ontologie“ stammt aus der Philosophie und befasst sich mit der Existenz von Dingen, wie diese beschrieben werden sollen und wie die Zuordnung unter allen anderen Dingen aussieht. In der Informatik wird die Ontologie dazu verwendet, um Modelle zu entwerfen , welche Typen, Eigenschaften und Beziehungen beschreiben, die für die jeweiligen Applikationen von Wichtigkeit sind [SET09, S. 127-128]. Damit möglichst viele Systeme solch eine Ontologie verstehen können, hat das W3C eine Spezifikation für die Beschreibungssprache Web Ontology Language (OWL) erstellt und als Standard[15] deklariert. Richtig hierbei ist, dass das Akronym OWL und nicht WOL lautet. Zurückzuführen ist dies auf Tim Finin[16], der hierdurch unter anderem weniger Probleme mit der Aussprache sah, da es wie das englische Wort Eule ausgesprochen wird. OWL gibt es in drei Ausführungen OWL Lite, OWL DL und OWL Full. Wie an den Namen zu vermuten ist, besitzt OWL Lite am wenigsten Ausdrücke, OWL FULL die meisten und OWL DL befindet sich dazwischen. Seit dem 27.10.2009 gibt es eine neuere Version für OWL[17], die informell auch OWL 2 genannt wird. Diese erweitert die bisherige Spezifikation beispielsweise um die Manchester Syntax[18]. Das Entwerfen einer Ontologie ist ein fortlaufender Prozess und für einen ersten Entwurf einer Ontologie bietet es sich an, mit OWL Lite zu beginnen, um den Umfang überschaubar zu halten. Auch kann die alten Spezifikation[19] von OWL verwendet werden, da diese mit OWL 2 kompatibel ist und die neue Funktionalität im ersten Schritt nicht benötigt wird.

[14] http://answers.semanticweb.com/questions/1638/jena-vs-sesame-is-there-a-serious-complete-up-to-date-unbiased-well-informed-side-by-side-comparison-between-the-two
[15] http://www.w3.org/TR/owl-ref/
[16] http://lists.w3.org/Archives/Public/www-webont-wg/2001Dec/0169.html
[17] http://www.w3.org/TR/owl2-overview/
[18] http://www.w3.org/TR/2012/NOTE-owl2-manchester-syntax-20121211/
[19] http://www.w3.org/TR/owl-semantics/

4.3.1 Modellierung der Ontologie

Zur Modellierung einer Ontologie stehen mehrere Alternativen für eine Syntax bereit. Als allererstes kann RDS/XML verwendet werden, da OWL auf dieser Syntax und dem RDF-Schema basiert. Wie in Abschnitt 4.1.5 bereits erwähnt, ist diese für den Menschen aber schwer lesbar. Alternativ kann Turtel als Syntax für die Erstellung der Ontologie verwendet werden, was sich anbietet, da bereits für die Graphen TriG verwendet wird und diese von Turtle abgeleitet ist.

Angefangen wird mit der Definition der Prefixe (siehe Listing 4.7) für die IRIs (Internationalized Resource Identifier). Dies erhöht, wie in Abschnitt 4.1.2 beschrieben, die Lesbarkeit deutlich, da nur noch die Abkürzung verwendet werden müssen und nicht die ganze IRI. Der Übersichtlichkeit halber wird nachfolgend der Ontologie ein leerer Präfix zugewiesen, später hat diese aber einen eigenen Präfix actions, um beim Zusammenführen mehrerer Ontologien eindeutig zu bleiben. Die weiteren Präfixe werden benötigt, um OWL(owl), das RDF-Schema(rdfs) und RDF/XML(rdf) verwenden zu können. Das XML Schema (xsd) wird benötigt, um die Datentypen definieren zu können.

```
1 @prefix : <http://plato.de/owl/actions/> .
2 @prefix owl: <http://www.w3.org/2002/07/owl#> .
3 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
4 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
5 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

Listing 4.7: Definition der Prefixe

In Listing 4.8 werden alle Klassen der RDF Ressourcen definiert, welche vom Typ `owl:Class` sind. Diese wiederum ist eine Unterklasse von `rdfs:Class`. Es gibt vier Basisklassen in dieser Ontologie:

- `:Action` (Maßnahme)
- `:Person` (allgemeine Person)
- `:Description` (allgemeine Beschreibung)
- `:Date` (allgemeines Datum)

```
1 :Action    rdf:type owl:Class .
2 :Person    rdf:type owl:Class .
3 :Description rdf:type owl:Class .
4 :Date    rdf:type owl:Class .
```

Listing 4.8: Definition der OWL Klassen

Bis auf Action besitzen alle anderen Klassen weitere Unterklassen (siehe Listing 4.9). Das sind die ersten Attribute die in Abschnitt 3.2.6 ermittelt wurden.

- `:Owner` (Verantwortlicher)
- `:Manager` (Manager)
- `:Creator` (Ersteller)
- `:ImplementationDescription` (Arbeitsanweisung)
- `:DefinitionDescription` (Umsetzungsbeschreibung)
- `:CreationDate` (Erstelldatum)
- `:StartDate` (Starttermin)
- `:EndDate` (Zieltermin)

```
:Owner  rdfs:subClassOf :Person .
:Manager rdfs:subClassOf :Person .
:Creator rdfs:subClassOf :Person .

:ImplementationDescription rdfs:subClassOf :Description
:DefinitionDescription rdfs:subClassOf :Description

:CreationDate rdfs:subClassOf :Date
:StartDate   rdfs:subClassOf :Date
:EndDate   rdfs:subClassOf :Date
```

Listing 4.9: Definition der Unterklassen

Danach werden alle Eigenschaften definiert (siehe Listing 4.10). Beispielsweise wird mit `:hasTitle rdf:type owl:DatatypeProperty` festgelegt, dass `:hasTitle` eine Eigenschaft ist, mit `rdfs:domain` für welche Klasse dies angewandt werden kann und mit `rdfs:range xsd:string`, dass diese vom Typ String ist. Alle anderen Eigenschaften werden genauso definiert, betreffen ggf. andere Klassen und sind von einem anderen Typ. Folgende Typen werden in dieser Ontologie benötigt:

- `xsd:string` (Text)
- `xsd:integer` (ganzzahliger Wert)
- `xsd:date` (Datum)

Mit dieser Definition werden die letzten Attribute für die Maßnahme festgelegt, die da wären:

- `:hasTitle` (Titel)
- `:hasPriority` (Priorität)
- `:hasKey` (Schlüssel)
- `:hasStatus` (Status)
- `:hasCategory` (Kategorie)
- `:hasImplementationRate` (Umsetzungsgrad)

```
:hasTitle rdf:type owl:DatatypeProperty ;
     rdfs:domain :Action ;
     rdfs:range xsd:string .

:hasName rdf:type owl:DatatypeProperty ;
    rdfs:domain :Person ;
    rdfs:range xsd:string .

:hasDescription rdf:type owl:DatatypeProperty ;
       rdfs:domain :Description ;
       rdfs:range xsd:string .

:hasDate rdf:type owl:DatatypeProperty ;
    rdfs:domain :Date ;
    rdfs:range xsd:date .

:hasPriority rdf:type owl:DatatypeProperty ;
       rdfs:domain :Action ;
       rdfs:range xsd:integer .

:hasKey rdf:type owl:DatatypeProperty ;
   rdfs:domain :Action ;
   rdfs:range xsd:string .
```

```
24
25 :hasStatus rdf:type owl:DatatypeProperty ;
26      rdfs:domain :Action ;
27      rdfs:range xsd:string .
28
29 :hasCategory rdf:type owl:DatatypeProperty ;
30        rdfs:domain :Action ;
31        rdfs:range xsd:string .
32
33 :hasImplementationRate rdf:type owl:DatatypeProperty ;
34            rdfs:domain :Action ;
35            rdfs:range xsd:integer .
```

Listing 4.10: Definition der Eigenschaften

Nun sollte auch klar werden wozu die Basisklassen mit den Unterklassen benötigt werden. Mit Hilfe der Basisklassen können Eigenschaften wie `:hasName`, `:hasDescription` und `:hasDate` einmal definiert werden und gleichzeitig in den Unterklassen verfügbar sein. So hat z.B. die Unterklasse :Owner auch die Eigenschaft `:hasName`, da ihre Basisklasse `:Person` ist. Für alle anderen Unterklassen verhält es sich genauso. In Abschnitt 4.3.2, in Abbildung 4.3 kann man diese Zusammenhänge sehr gut erkennen.

Zu guter letzte müssen noch die Relationen der Ontologie bestimmt werden (siehe Listing 4.11).

```
1 :Action :createdBy :Creator .
2 :Action :managedBy :Manager .
3 :Action :ownedBy  :Owner .
4 :Action :describedBy :DefinitionDescription .
5 :Action :documentedBy :ImplementationDescription .
6 :Action :createdOn :CreationDate .
7 :Action :startsOn :StartDate .
8 :Action :endsOn  :EndDate .
```

Listing 4.11: Definition der Unterklassen

Alle diese Relationen haben gemeinsam, dass sie die Maßnahme mit einer der Unterklassen verbinden.

- `:createdBy` (erstellt durch)
- `:managedBy` (verwaltet durch)
- `:ownedBy` (besessen durch)
- `:describedBy` (beschrieben durch)
- `:documentedBy` (dokumentiert durch)
- `:createdOn` (erstellt am)
- `:startsOn` (startet am)
- `:endsOn` (endet am)

Mit diesem ersten Entwurf wird eine Maßnahme rudimentär beschrieben und im Laufe der Zeit kann diese immer wieder ergänzt und erweitert werden, um eine Maßnahme mit dieser Ontologie bestmöglich darzustellen.

4.3.2 Ontologie (grafisch)

In Abbildung 4.3 wird die in Abschnitt 4.3.1 vorgestellte Ontologie, einer Maßnahme, grafisch dargestellt.

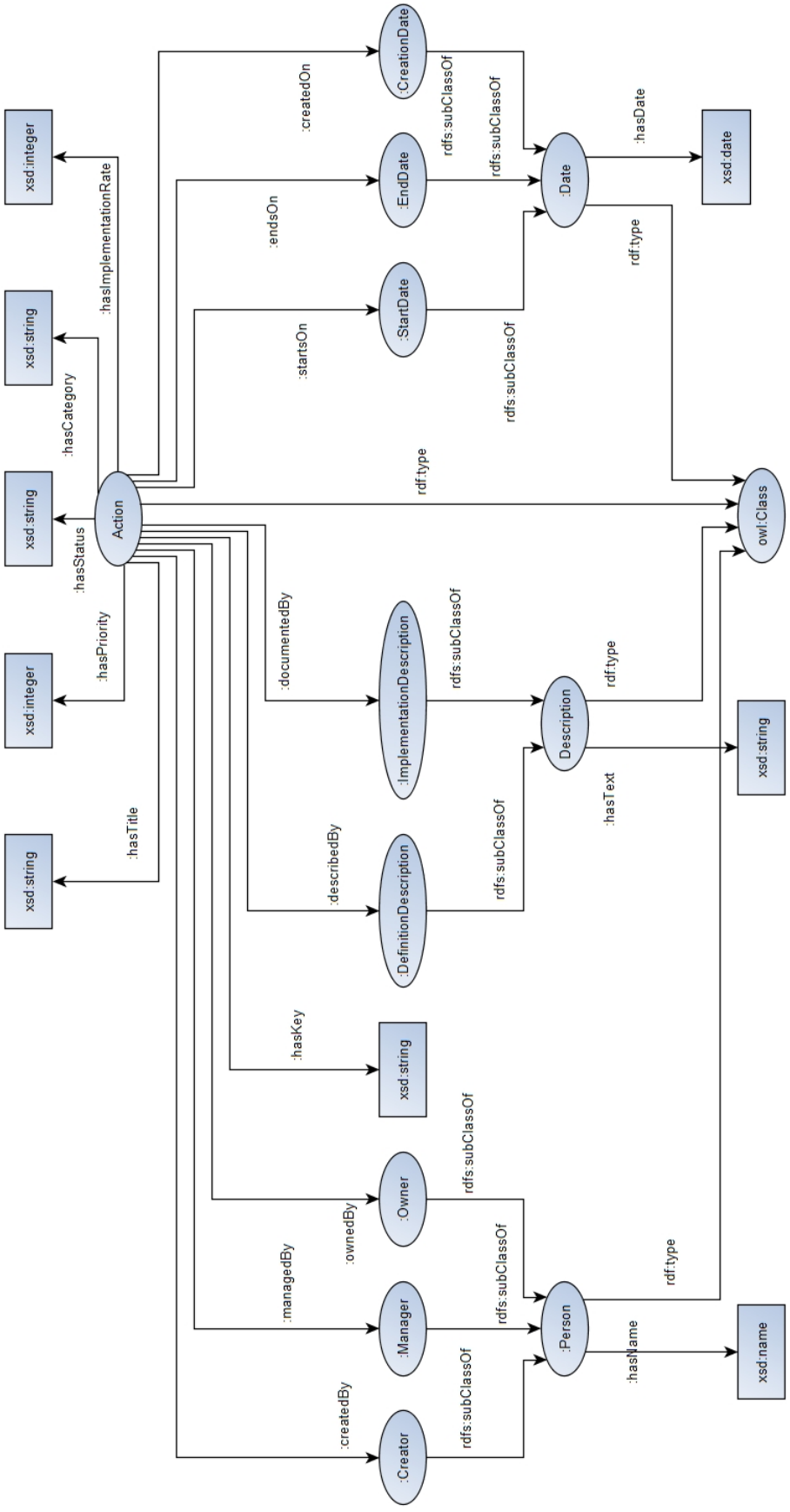

Abbildung 4.3: Grafische Maßnahmenontologie

5 Implementierung in e1ns.actions

Als Triplestore wurde sich in Abschnitt 4.2.5 zwar für Apache Jena entschieden, für das erstellen der Graphen wird jedoch RDFLib verwendet, da dieses wie alle PLATO Produkte in Python (siehe R01) und nicht wie Apache Jena in Java geschrieben ist.

5.1 pyseki

Um mit Fuseki arbeiten zu können, wurden anhand von dem Klassendiagramm aus Abbildung 5.1 das Modul `pyseki` implementiert. Dieses beinhaltet die folgenden fünf Klassen:

- `FusekiServer`
- `FusekiConnection`
- `FusekiResult`
- `FusekiResultRow`
- `ConnectionError`

In den folgenden Abschnitten wird detailliert auf diese eingegangen.

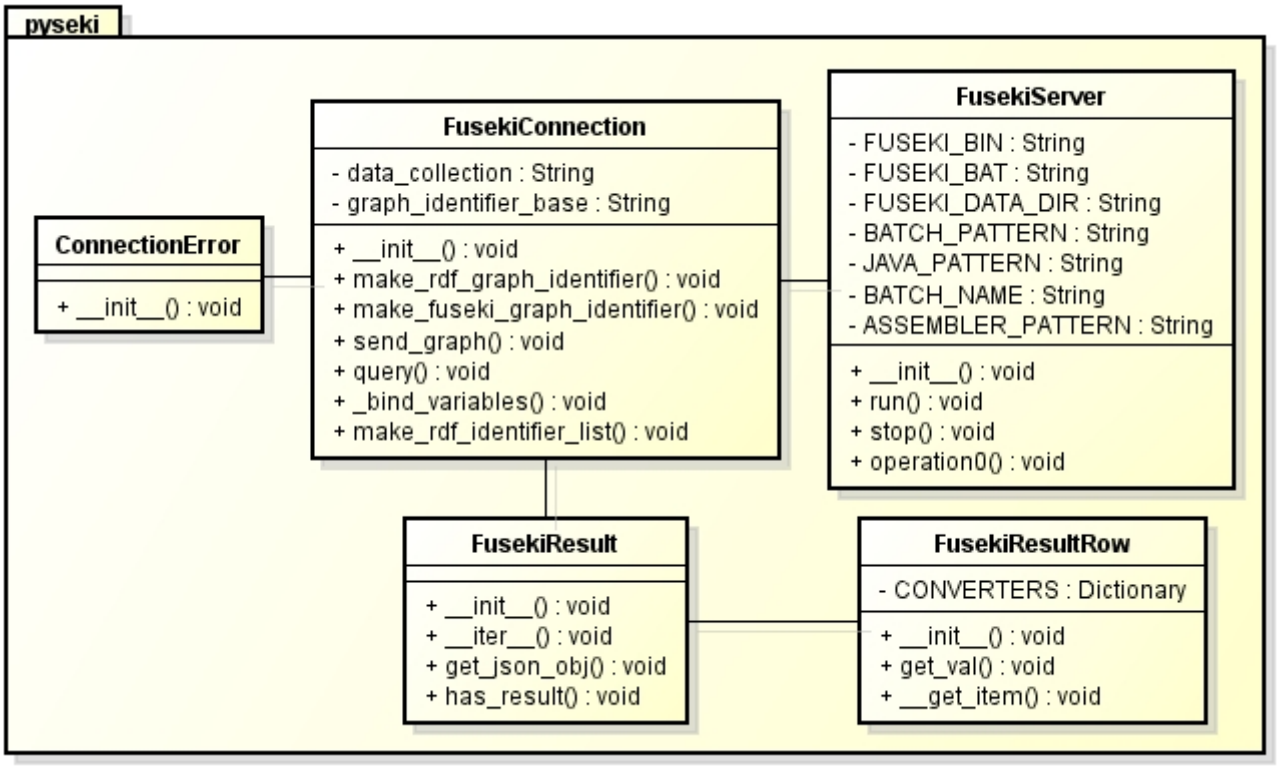

Abbildung 5.1: Klassendiagramm vom Modul pyseki

5.1.1 FusekiServer

Die Klasse `FusekiServer` kümmert sich um das Starten und Beenden des Fuseki Prozesses und ist als Singelton implementiert, damit sichergestellt ist, dass es immer nur eine Instanz zu Fuseki gibt. Sie verfügt über die drei Funktionen `__init__`, `run` und `stop`.

5.1.1.1 Parameter

FUSEKI_BIN Gibt den Namen, des Ordners, der Fuseki Installation, im Bin-Ordner der Applikation an.

FUSEKI_BAT Name der Batch-Datei, zum starten des Fuseki Servers

FUSEKI_DATA_DIR Pfad zur Triplestore Datenbank

BATCH_PATTERN Dieser Parameter dient als Template, welches mit den anderen Parametern gefüllt wird, um den Inhalt der Batch-Datei zu erzeugen.

JAVA_PATTERN Informationen zum starten von Fuseki über Java

BATCH_NAME Batch-Dateiname für die erzeugte Batch-Datei

ASSEMBLER_PATTERN Assembler Spezifikation. Eine genaue Beschreibung von Jenas Assembler findet man unter `https://jena.apache.org/documentation/assembler/`.

5.1.1.2 __init__

Bei dieser Funktion handelt es sich um den Konstruktor, der Konfigurationen ausliest, wie z.B. wo Fuseki installiert ist und auf welchem Port dieser gestartet werden soll.

5.1.1.3 run

Die Funktion `run` startet einen Prozess von Fuseki. Dies ist auf Grund der Tatsache nötig, dass Apache Jena in JAVA implementiert ist und somit nicht einfach in die Applikation integriert werden kann. Fuseki kann dann von e1ns.actions aus über HTTP[1] (zur Übertragung von Daten über ein Netzwerk) angesprochen werden.

5.1.1.4 stop

Mit Hilfe der Funktion `stop` kann der Fuseki Prozess wieder beendet werden.

5.1.1.5 Beispiel

Der `FusekiServer` kann beispielsweise über die Konsole aus einer Pythondatei heraus gestartet werden (siehe Listing 5.1).

```
1 import atexit
2 import sys
3
4 from plato.eplm.pyseki import FusekiServer
5
6 FUSEKI = None
```

[1] Hypertext Transfer Protocol

```
7
8  def end_fuseki():
9    if FUSEKI is not None:
10     FUSEKI.stop()
11
12 # Register cleanup function with python (stop fuseki)
13 atexit.register(end_fuseki)
14
15 def main():
16   global FUSEKI
17   # Init fuseki
18   FUSEKI = FusekiServer(blocking=True, execute=True, start_batch=False)
19   try:
20     # Start fuseki
21     FUSEKI.run()
22   except KeyboardInterrupt:
23     # Stop fuseki
24     FUSEKI.stop()
25     sys.exit()
26
27 if __name__ == "__main__":
28   main()
```

Listing 5.1: Start des Fuseki Servers

5.1.2 FusekiConnection

Mit der Hilfe der Klasse `FusekiConnection`, werden die Graphen an den Triplestore geschickt und sind über Anfragen abfragbar.

5.1.2.1 ___init___

Der Konstruktor von `FusekiConnection` erzeugt oder holt sich eine Instanz von `FusekiServer`. Durch die Implementierung als Singelton wird sichergestellt, dass nur eine Instanz erzeugt wird und immer wieder auf diese zugegriffen werden kann.

5.1.2.2 send_graph

`send_graph` bekommt ein RDFLib Graphobjekt übergeben, welches in das entsprechende Format serialisiert wird und per HTTP Request an Fuseki und somit Apache Jena geschickt wird.

5.1.2.3 query

Wie der Name es schon sagt, kann man mit Hilfe von `query` Anfragen an Apache Jena, über Fuseki schicken. Als Abfragesprache wird der vom W3C vorgeschlagene Standard SPARQL verwendet (siehe R11) und da die Graphen als named graphs gespeichert werden, können diese über Anfragen gezielt angesprochen werden.

Die Funktion `query` von `FusekiConnection` verfügt über die Parameter `sparql_select`, `from_list`, `sparql_where`, `binding_dict` und `output`.

```
def query(self, sparql_select, from_list, sparql_where, binding_dict, output):
  ...
```

Listing 5.2: Parameter für query

Bei `sparql_select` handelt es sich um die Anfrage in SPARQL. Für `sparql_select` kann auch eine Anfragevorlage verwendet werden, die nur mit Daten gefüllt werden muss. Diese Vorlage ist von ist ein Template, die die Daten aus `binding_dict` bekommt und daraus eine SPARQL Anfrage erstellt, die Apache Jena verarbeiten kann.

```
  if binding_dict is not None:
    sparql_select = self._bind_variables(sparql_select, binding_dict)

  if isinstance(sparql_select, Template):
    sparql_select = sparql_select.template
```

Listing 5.3: SPARQL Anfrage und Anfragevorlage

In der `from_list` können ein oder mehrere Graphenschlüssel übergeben werden, die in Fuseki Graph Identifier umgewandelt werden.

```
  for graph_id in from_list:
    sparql_from_list.append("from <%s>" % self.make_fuseki_graph_identifier(graph_id))
```

Listing 5.4: Fuseki Graph Identifier

Das `Where` Statement kann über `sparql_where` übergeben werden oder kann schon Teil von `sparql_select` sein.

```
  if sparql_where is None:
    sparql_comp = sparql_select.split("where")
    sparql_select = sparql_comp[0]
    sparql_where = "where %s" % sparql_comp[1]

  sparql = """
    PREFIX rdf: <%s>
    PREFIX xsd: <%s>
    PREFIX actions: <%s>
    BASE <%s>
    %s
    %s
    %s
    """ % (rdflib.RDF, rdflib.XSD, ACTIONS_URI, self.graph_identifier_base, sparql_select
        , "\n".join(sparql_from_list), sparql_where)

  url = "%s/query" % self.end_point_coll

  logDebug("Fuseki query = %s" % sparql)
```

Listing 5.5: SPARQL where

Im letzten Teil von `query` sendet einen Request mit der SPARQL Anfrage an Fuseki. Mit output wird angegeben, ob das Resultat als JSON[2] oder in reiner Textform zurückgegeben werden soll.

```
  try:
    r = self.session.post(url, data=dict(query=sparql, output=output))
    if r.status_code != 200:
```

[2] JavaScript Object Notation

```
4       raise Exception("query - Fuseki error'%s'" % r.text)
5     except requests.exceptions.ConnectionError, e:
6       msg = "Fuseki could not be reached at <%s>" % self.end_point_base
7       logException(msg)
8       if output == "json":
9         return [{"": tools.makePwsFaultDict(msg)}]
10      else:
11        raise ConnectionError(e, msg)
12
13    if output == "json":
14      return FusekiResult(r.json())
15    else:
16      return r.text
```

Listing 5.6: SPARQL where

5.1.2.4 make_fuseki_graph_identifier

`make_fuseki_graph_identifier` erstellt die URIs für Graphen.

5.1.2.5 make_rdf_graph_identifier

`make_rdf_graph_identifier` erstellt die URIs der einzelnen Resourcen.

5.1.2.6 make_rdf_identifier_list

`make_rdf_identifier_list` macht aus einer Liste von Schlüsseln, eine Liste von URIs.

5.1.2.7 _bind_variables

`_bind_variables` wird verwendet wenn eine SPARQL Anfragevorlage verwendet wird (siehe Abschnitt 5.1.2.3). Es fügt hierzu die Daten aus `binding_dict` in die Vorlage ein.

5.1.3 FusekiResult

`FusekiResult` stellt ein Objekt für das Ergebnis aus Fuseki bereit, welches die Daten ausliest und zuzätliche Hilfsfunktionen, wie `get_json_obj` und `has_result` bereitstellt.

5.1.3.1 __init__

Im Konstruktor von `FusekiResult` wird die Rückgabe von Fuseki ausgelesen.

5.1.3.2 __iter__

In Python wird die Funktion `__iter__` genutzt, wenn ein Objekt iteriert wird. Diese wird in FusekiResult überschrieben, sodass die einzelnen Zeilen des Ergebnisses als FusekiResultRow Objekt zurückgeliefert werden.

5.1.3.3 get_json_obj

Diese Funktion liefert das Resultat als JSON Objekt zurück.

5.1.3.4 has_result

Durch `has_result` lässt sich prüfen ob Fuseki ein Ergebnis zurückgeliefert hat.

5.1.4 FusekiResultRow

5.1.4.1 __init__

Der Konstruktur setzt die Zeile als Objektattribut.

5.1.4.2 get_val

Um an den Wert einer Zeile zu gelangen verwendet man `get_val`. Benötigt wird der Variablenname und optional ein Umwandlungstyp, für den Wert.

5.1.4.3 __getitem__

Wird von Python aufgerufen, wenn ein Objekt nach dem Muster `Object[key]` aufgerufen wird. Für `FusekiResultRow` wird sie überschrieben, sodass sie den Typen ermittelt und `get_val` aufruft.

5.1.5 ConnectionError

Der `ConnectionError` ist ein spezieller Fehler, der geworfen werden kann, wenn ein Verbindungsfehler mit Fuseki auftritt.

5.2 Triplestore Provider

In Abbildung 5.2ist das Klassendiagramm mit den Klassen `TriplestoreProvider` und `ElementGraphConverter` sowie die Verbindungen zum Modul `pyseki` (siehe Abbildung 5.1) und dem Modul `ActionsWebApp`, mit der Implementierung von e1ns.actions.

5.2.1 __init__

Alle Daten von e1ns.actions werden z.Z. in einer relationalen Datenbank gespeichert, somit ist es nötig dem Konstruktor beim initialisieren vom TriplestoreProvider eine Instanz vom ActionsProvider zu geben, damit dieser auf die Daten zugreifen kann. Weitergehend initialisiert er selber den ElementGraphConverter um die Daten aus e1ns.action in Graphen umzuwandeln und später beim Import von RDF Dateien, die Daten wieder in der relationalen Datenbank zu speichern.

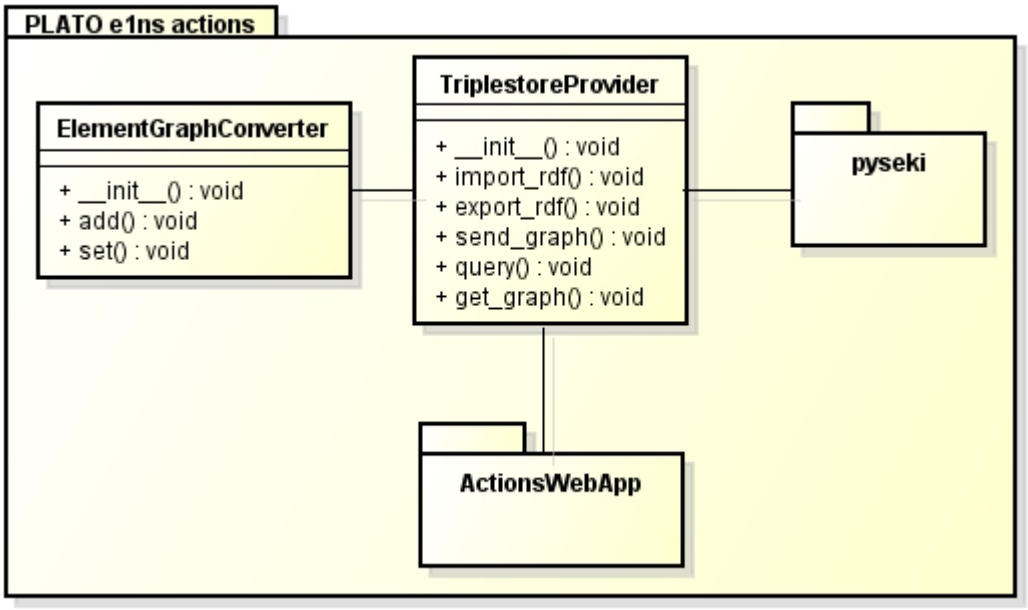

Abbildung 5.2: Klassendiagramm TriplestoreProvider

5.2.2 export_rdf

Für den Export einer RDF Datei, wird Fuseki und somit Apache Jena z.Z. nicht benötigt, da die Daten wie bereits beschrieben, in einer relationalen Datenbank liegen. Es wird mit Hilfe von RDFLib ein Graph aus den Daten der relationalen Datenbank erzeugt, in das entsprechende Format serialisiert und als RDF Datei zum herunterladen angeboten. Hierbei ist das Erstellen des Graphen in eine extra Funktion ausgelagert, sodass er auch für den Export in den Triplestore verwendet werden kann.

5.2.3 get_graph

`get_graph` ist die erwähnte Funktion, die die Daten aus e1ns.actions, anhand des Schlüssels des Elements holt und mit Hilfe vom `ElementGraphConverter einen Graphobjekt erzeugt.`

```
1  def get_graph(self, element_key, element_type):
2    supported_element_list = ["scio.project", ]
3
4    if element_type not in supported_element_list:
5      raise ValueError("Element type '%s' is not supported. Following element types can be
           used: %s" % (element_type, ", ".join(supported_element_list)))
6
7    # Creating graph
8    element_graph_key = FusekiConnection.make_rdf_graph_identifier(element_key,
         element_type)
9    graph = rdflib.Graph(identifier=element_graph_key)
10
11   # Get root element from e1ns.actions
12   root_element = self.actions_provider.get_element(element_key, element_type)
13
14   # Add root element to graph
15   root_element_node = self.converter.add(root_element, graph)
16
17   # Get all direct children that have the type actions.action
18   child_gen = self.actions_provider.get_children_by_type(element_key, element_type, "
         actions.action")
```

```
19
20   for action_element in child_gen:
21     action_node = self.converter.add(action_element, graph)
22     graph.add((root_element_node, ACTIONS["project_actions"], action_node))
23
24   return graph
```

Listing 5.7: Export als RDF Datei

5.2.4 import_rdf

Der Import erhält eine RDF Datei, die mit RDFLib ausgelesen und in ein Graphobjekt umgewandelt wird. Aus dem Graphen wird ein Dictionatry[3] erstellt und an den `ElementGraphConverter` übergeben, der die Daten in der relationalen Datenbank speichert. Als Rückgabe liefert diese Funktion ein JSON Objekt mit dem Status zurück, mit dem überprüft werden kann, ob ggf. ein Fehler aufgetreten ist.

5.2.5 send_graph

Wie in Abschnitt 5.2.2 wird zum Erstellen des Graphen `get_graph` verwendet. Da die Graphen in e1ns.actions alle ein named graph sind, kann dieser direkt über die `FusekiConnection` (siehe Abschnitt 5.1.2) an den Triplestore geschickt werden. Zurückgeliefert wird wie bei `import_rdf` ein JSON Objekt.

5.2.6 query

Für die Anfrage an den Graphen benötigt man zum einen den Graphenschlüssel(`graph_key`) und zum anderen die eigentliche Anfrage (`query`). Mit diesen beiden Parametern kann eine Anfrage, über die `FusekiConnections` abgesetzt werden.

```
1 def query(self, graph_key, query):
2   fuseki_conn = FusekiConnection()
3   return fuseki_conn.query(query, from_list=[project_graph_key])
```

Listing 5.8: Abfrage des Triplestores

5.3 ElementGraphConverter

Wie in Abschnitt 5.2.1 beschrieben wird die Klasse ElementGraphConverter dazu verwendet Daten aus der Datenbank von e1ns.actions in Graphen umzuwandeln (`add`) und importierte Daten in der selben Datenbank zu speichern(`set`).

5.3.1 add

Die Funktion `add` erhält ein Elementobjekt, dass aus e1ns.actions stammt und ein Graphobjekt, in das die Daten geschrieben werden sollen. `add` ruft dann, je nach Elementtyp, eine entsprechende Funktion auf, die dem Muster `add_` plus dem Elementtypen, folgt. Die Funktion für Maßnahmen lautet also `add_actions_action` und für Projekte aus SCIO™`add_scio_projekt`. In diesen Funktionen werden die Elemente typspezifisch behandelt. Mit diesem Konzept, lassen sich sehr leicht weitere Elemente hinzufügen.

[3] Python Datentyp, in dem Daten über einen Schlüssel abfragbar sind.

5.3.2 set

`set` verhält sich genau wie `add`, mit der Ausnahme, dass kein Graph erzeugt wird, sondern in die Datenbank von e1ns.actions geschrieben wird. Die Funktion für Maßnahmen lautet hier `set_actions_action`.

5.4 Zusammenfassung der Implementierung

In Abbildung 5.3 sind die Zusammenhänge der einzelnen Implementierungsteile zu sehen. `pyseki` ist für die Verbindung und Kommunikation mit Fuseki (Apache Jena) verantwortlich und ist mit dem `TriplestoreProvider` verbunden. Dieser verarbeitet SPARQL Anfragen, schickt die Graphen an den Triplestore, liest RDF Dateien aus und exportiert auch solche. Der `TriplestoreProvider` wiederrum, ist mit `ActionsWebApp` verbunden, um die importierten Daten in der gesonderten Datenbank von e1ns.actions speichern zu können. Die `ActionsWebApp` ist die Implementierung von e1ns.actions und stellt die Schnittstelle nach außen, zu externen Systemen (z.B. für Systeme aus Abschnitt 3.2) dar, aber auch zu den weiteren Produkten von PLATO, wie SCIO™. Hierfür stellt SCIO ein API namens SAPI (`SCIO API`) zur Verfügung.

Möchten Kunden mehr als nur einen Dateiaustausch, müsste im Zuge eines Projektes, nur noch ein Connector geschrieben werden, der die jeweiligen Schnittstellen miteinander verbindet.

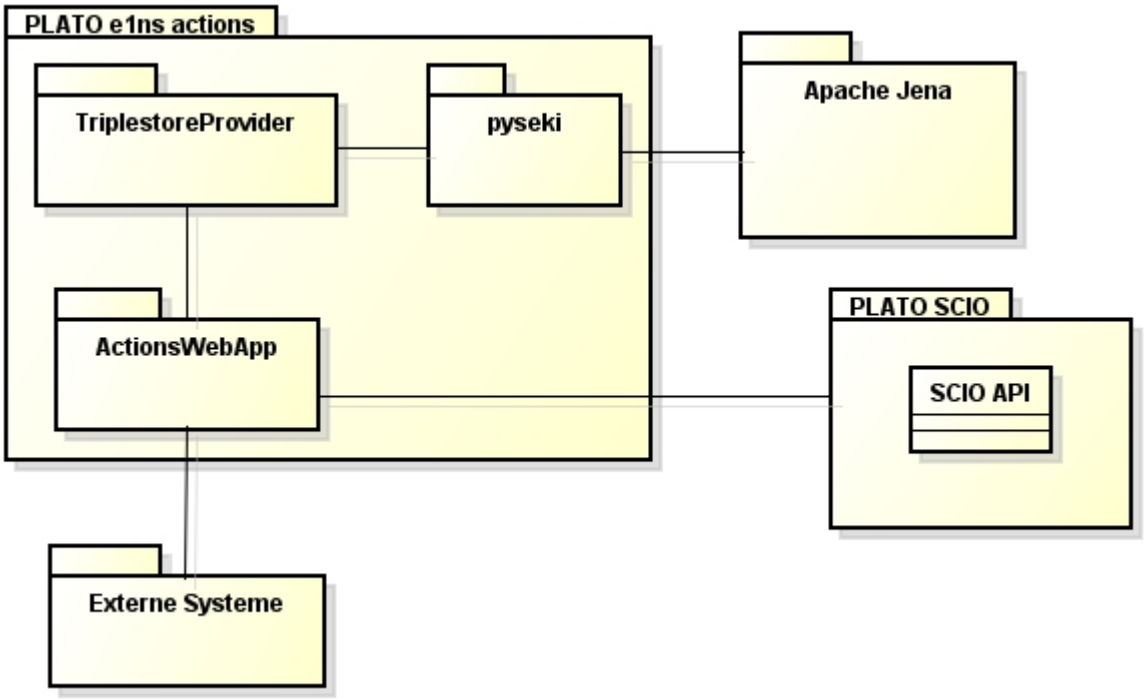

Abbildung 5.3: Übersicht der Zusammenhänge

5.5 Nachweisführung

In diesem Kapitel werden die Implementierung und die einzelnen Funktionalitäten getestet, um die Funktionsfähigkeit sicherzustellen.

5.5.1 Unittest

Pythons Standardbibliothek stellt das Modul `unittest` zur Verfügung, welches die Funktionalität des Moduls JUnit implementiert, dem De-facto-Standard zur testgetriebenen Entwicklung in Java [KE07, S. 602]. Bei testgetriebener Entwicklung werden die Testfälle vor der eigentlichen Entwicklung geschrieben. Es wird festgelegt, welche Parameter übergeben werden sollen und was die Funktion letztendlich als Rückgabewert zurückliefert. Während der Entwicklung können jederzeit diese Unittests durchgeführt werden, um zu prüfen, ob die Funktion den Spezifikationen entspricht und korrekte Resultate liefert. Eine weitere Möglichkeit, eine Funktion zu testen, ist der Negativtest. Hierbei wird geprüft, wie die Funktion mit falschen Daten umgeht. Diese sollten von der Funktion natürlich abgefangen werden, ggf. entsprechende Fehlermeldungen generieren und nicht die Applikation z.B. Abstürzen oder fehlerhafte Resultate liefern. Aber nicht nur während der Entwicklung, sondern auch danach können die Unittest automatisiert ausgeführt werden und prüfen, ob durch nachträgliche Änderungen im Programmcode, die Funktion weiterhin wie gewünscht funktioniert.

Eine Eigenschaft von `unittest` ist, das mit `setUpModule` und `tearDownModule`, festgelegt werden kann, was vor und nach dem Unittest ausgeführt werden soll. Alle Webapplikationen von PLATO verfügen über die Klassen `TestServer`, die eine PLATO Webapplikation starten kann. Mit den Funktionen kann also, e1ns.actions vor dem Unittest gestartet werden und nach dem Unittest wieder beendet werden, ohne das manuell ein Prozess gestartet werden muss.

```
1 server = TestServer(start_script=getCommonPath(COMMONPATH_Project, "actions", "
      plato_e1ns_actions.py"))
2
3 def setUpModule():
4   server.start()
5
6 def tearDownModule():
7   server.stop()
```

Listing 5.9: Abfrage des Triplestores

Weitergehend gibt es die Klasse `PwsBrowserSession`, die mit Hilfe des Python Moduls `urllib2` Anfragen über HTTP an e1ns.actions absetzen kann und für die folgenden Test verwendet wurde.

5.5.2 Fuseki

Als erstes muss sichergestellt werden, dass die Anbindung an Fuseki gestartet und beendet werden kann. Hierzu wird die in Abschnitt 5.1.1.5 vorgestellte Pythondatei gestartet, geprüft, ob der Prozess gestartet wird und ob durch das Werfen eines KeyboardInterrupt der Prozess beendet wird.

Test-Nr.	Beschreibung des Testfalles	Testresultat
F01	Fusekiprozess starten	ok
F02	Fusekiprozess beenden	ok

Tabelle 5.1: Testübersicht für Fuseki

5.5.3 Import und Export von RDF Dateien

Im nächsten Schritt muss das Importieren und das Exportieren von RDF Dateien überprüft werden. Dazu werden RDF Dateien in den verschiedenen Formaten erstellt und importiert. Durch die in Abschnitt 5.2.4 beschriebenen Rückgaben in JSON, kann überprüft werden, ob der Import erfolgreich war oder ob es zu einem Fehler kam. Diese importierten Daten können darauf exportiert werden und die ausgegebenen RDF Dateien, mit denen für den Import bereitgestellten Dateien verglichen werden.

Test-Nr.	Beschreibung des Testfalles	Testresultat
RDF01	RDF Dateiimport (N-Triples)	ok
RDF02	RDF Dateiimport (N3)	ok
RDF03	RDF Dateiimport (Turtle)	ok
RDF04	RDF Dateiimport (TriG)	ok
RDF05	RDF Dateiimport (RDF/XML)	ok
RDF06	RDF Dateiimport (RDFa)	ok
RDF07	RDF Dateiexport (N-Triples)	ok
RDF08	RDF Dateiexport (N3)	ok
RDF09	RDF Dateiexport (Turtle)	ok
RDF10	RDF Dateiexport (TriG)	ok
RDF11	RDF Dateiexport (RDF/XML)	ok
RDF12	RDF Dateiexport (RDFa)	ok

Tabelle 5.2: Testübersicht für Import und Export von RDF Dateien

5.5.4 Triplestore

Für den Import in den Triplestore werden die zuvor nach e1ns.actions importierten Daten verwendet, um die Funktionsfähigkeit prüfbar zu machen. Wie der Import der RDF Dateien liefert der Import in den Triplestore eine Rückgabe in JSON, die den Erfolg oder Misserfolg (Fehler) des Imports zurück liefert.

Test-Nr.	Beschreibung des Testfalles	Testresultat
T01	Import eines Graphen in den Triplestore	ok

Tabelle 5.3: Testübersicht für den Import in den Triplestore

Als letzten Schritt muss das Abfragen von Graphen im Triplestore geprüft werden. Hierfür werden SPARQL Anfragen, wie in Listing 5.10 erstellt.

```
1 select ?action_title
2   where {
3     ?action_node actions:owner "Rayner Vesely"^^xsd:Name .
4     ?action_node actions:title ?action_title .
5   }
```

Listing 5.10: SPARQL Anfrage

Diese Anfrage holt sich alle Maßnahmetitel (`action_title`) für Maßnahmen mit dem Verantwortlichen (`actions:owner`) „Rayner Vesely“. Da die importierten Daten bekannt sind, können die Resultate darauf geprüft werden.

Test-Nr.	Beschreibung des Testfalles	Testresultat
T02	Maßnahmetitel für einen bestimmten Verantwortlichen	ok

Tabelle 5.4: Testübersicht für Anfragen an den Triplestore

5.6 Zusammenfassung und Ausblick

In diesem Kapitel wird die Bachelorarbeit zusammenfassend erläutert sowie ein Ausblick auf mögliche nächste Schritte, die dieser Arbeit folgen könnten, aufgezeigt.

5.6.1 Zusammenfassung

Ziel dieser Bachelorarbeit ist es, die Maßnahmenapplikation e1ns.actions so zu erweitern, dass diese Informationen aus dem Semantischen Web verarbeiten und behandeln kann. Für diesen Zweck wurden in Abschnitt 3.2 mehrere Maßnahmensysteme untersucht und ein Standardmodell für Maßnahmen entwickelt. Mithilfe der in Abschnitt 3.1 vorgestellten Anforderungen wurde die Beschreibungssprache TriG (siehe Abschnitt 4.1) und der Triplestore (siehe Abschnitt 4.2) Apache Jena ausgewählt. Darauffolgend konnte in Abschnitt 4.3.1 durch das Standardmodell und der Beschreibungssprache eine Ontologie entworfen werden, die eine Maßnahme im semantischen Web darstellt.

Alle diese Grundlagen dienten der Umsetzung aus Kapitel 5. Es wurde Wert darauf gelegt, einen objektorientierten modularen Ansatz zu wählen, so dass e1ns.actions in Zukunft weiterhin gut erweiterbar ist.

Unter Verwendung der Unittests (siehe Abschnitt 5.5), wurde schlussendlich die Funktionsfähigkeit sichergestellt. Zudem wurden Anforderungen wie R05, die Unterstützung mehrerer Beschreibungssprachen von RDF Dateien, in Abschnitt 5.5.3, durch den Import und Export von RDF Dateien in verschiedenen Formaten überprüft.

Abschließend lässt sich sagen, dass sich durch die Erweiterung von e1ns.actions um das Semantische Web, nun eine Vielzahl von Möglichkeiten eröffnen. Folgende Vorteile sind zu nennen:

- Bündelung aller Maßnahmen aus mehreren Systemen an einer Stelle
- Beliebige Vernetzung von Maßnahmendaten
- Abfragen über alle Maßnahmen erfolgen zentral

Mit diesen Argumenten kann PLATO jetzt an Kunden herantreten, die sich bisher gesträubt haben ein weiteres Maßnahmensystem einzuführen.

5.6.2 Ausblick

Bereits jetzt ist e1ns.actions in der Lage, SPARQL Anfragen für Maßnahmen entgegenzunehmen. Zukünftig sollen, wie in der Einleitung beschrieben, verschiedene Ontologien vernetzt werden und es, soll möglich sein, alle Daten der PLATO Produkte sowie anderer Systeme, die keine Maßnahmen darstellen, aber beispielsweise über Projekte verknüpft sind, zu vereinen. Der Grundstein hierfür ist durch die named graphs bereits gelegt und durch den objektorientierten Ansatz der Implementierung leicht zu erweitern.

Durch die Einführung eines Triplestores gibt es neben diesem die bisherige relationale Datenbank, die die Daten für e1ns.actions vorhält. In der Zukunft strebt PLATO an vollständig auf den Triplestore über zu gehen und somit die Daten nicht doppelt pflegen zu müssen.

PLATO führt Testautomatisierung nach der Methode des „Fuzzings“[SGA07] durch, welches durch Ontologie nun gut ergänzt werden kann.

Literaturverzeichnis

[Brü14] Brügger, Stephan: *Arbeitszeugnis für Rayner Vesely*. 2014

[HKRS08] Hitzler, Pascal ; Krötzsch, Marcus ; Rudolph, Sebastian ; Sure, York: *Semantic Web*. Springer-Verlag, 2008

[HS05] Hölzer, Michael ; Schramm, Michael: *Qualitätsmanagement mit SAP*. SAP PRESS, 2005

[KE07] Kaiser, Peter ; Ernesti, Johannes: *Python*. Galileo Computing, 2007

[MN04] Meiners, Johannes ; Nüsser, Wilhelm: *SAP Interface Programming*. SAP PRESS, 2004

[O'R09] O'Reilly, Tim: *What Is Web 2.0*. O'Reilly Media, 2009

[SET09] Segaran, Toby ; Evans, Colin ; Taylor, Jamie: *Programming the Semantic Web*. O'Reilly Media, 2009

[SGA07] Sutton, Michael ; Greene, Adam ; Amini, Pedram: *Fuzzing*. Pearson Education, 2007