Universität Bremen

Fachbereich 3: Mathematik und Informatik

Bachelorarbeit

Visualisierung von Ontologien zur Unterstützung von Modulextraktion

Jan Kleinekathöfer

Matrikel-Nr. 432 462 8

5. Dezember 2019

1. Gutachter: Prof. Dr. Thomas Schneider
2. Gutachter: Dr. Serge Autexier
Betreuer: Prof. Dr. Thomas Schneider

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Bremen, den 5. Dezember 2019

_____

Jan Kleinekathöfer

Zusammenfassung

Um komplexe Aufgaben durchzuführen, ist ein Mensch in der Lage, auf abstraktes Wissen zurückzugreifen, welches er/sie sich über einen langen Zeitraum angeeignet hat. Damit solche Aufgaben auch automatisiert durchgeführt werden können, muss dieses Wissen für Maschinen verständlich repräsentiert werden. Diese Möglichkeit bietet das Konzept der Ontologie. Da für unterschiedliche Aufgaben unterschiedliches Wissen benötigt wird, werden auch verschiedene Ontologien verwendet. Einige Aufgaben überschneiden sich jedoch in ihrem benötigten Wissen. Um in einer Ontologie Wissen aus einer anderen wiederverwenden zu können, ohne das gesamte Wissen der Ontologie zu übernehmen, gibt es das Konzept des Moduls. Für ein Modul kann spezifiziert werden, welches Wissen enthalten sein soll. Für viele Modularten ist das Extrahieren des Moduls anhand einer Spezifikation automatisch durchführbar, die Spezifikation des Moduls muss jedoch meist manuell durchgeführt werden. Damit die Anwendenden in der Lage sind, diese Spezifikation durchzuführen, benötigen sie Informationen über die Ontologie.

Wie Visualisierungen diese Informationen bereitstellen können, wird in dieser Arbeit untersucht. Dabei wird auf unterschiedliche Methoden, ein Modul zu spezifizieren, eingegangen. Für die Spezifikation über Signaturen erweist sich eine Visualisierung bestehend aus einer eingerückten Liste und einem Graphen als vielversprechend. Bei Verfahren, welche die Dekomposition einer Ontologie verwenden, wird gezeigt, dass eine abgewandelte Form der atomaren Dekomposition für die Modulextraktion einsetzbar ist. Außerdem wird eine Anwendung zur Modulextraktion basierend auf der erweiterten atomaren Dekomposition implementiert. Diese Anwendung zeigt, dass die Modulextraktion basierend auf atomarer Dekomposition für Ontologien begrenzter Größe durchführbar ist.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Wissensrepräsentation spielt in vielen Bereichen der Informatik eine zentrale Rolle. Um automatisierte Systeme umsetzen zu können, wird ein Weg benötigt, Wissen für diese Systeme nutzbar zu machen. Dies kann geschehen, indem bei der Erstellung des Systems implizit Wissen in dessen Quellcode eingebaut wird. Für Anwendungen, welche mit größeren Mengen an Wissen umgehen müssen, kann jedoch eine explizite Formulierung dessen sinnvoll sein.(vgl. [Mus14]) Daher werden explizite Formen der Wissensrepräsentation in der Robotik, für intelligente Systeme und im Zusammenhang mit dem Semantic Web sowie vielen anderen Anwendungsfeldern eingesetzt.

Eine Technologie, welche sich zur Repräsentation von Wissen durchgesetzt hat, ist das Konzept der Ontologie. Ontologien sind in der Lage, Wissen auf eine Weise zu repräsentieren, welche es Maschinen ermöglicht, diese ohne großen Aufwand zu nutzen. Neben der Repräsentation von Wissen können Ontologien auch genutzt werden, um aus bestehendem Wissen Neues zu schließen. Zum Verfassen einer Ontologie bedarf es einer formalen Sprache. Dabei ist es üblich, das in einer Ontologie enthaltene Wissen in einzelnen Aussagen zu formulieren. Um Wissen über ein umfangreiches Gebiet zu repräsentieren, müssen viele dieser Aussagen formuliert werden.

Damit gleiches Wissen nicht wiederholt formuliert werden muss, möchte man Aussagen aus Ontologien wiederverwenden können. Bei der Umsetzung dieses Prozesses können Module verwendet werden, welche in anderen Anwendungsfällen eingesetzt werden. Ein Modul ist eine Ontologie. In dieser Arbeit wird der Begriff des Moduls für eine Teilmenge der Aussagen einer ursprünglichen Ontologie verwendet. Eine Teilmenge an Aussagen für ein Modul zu wählen, welche das gewünschte Wissen enthält, ist dabei eine nicht triviale Aufgabe. Problematisch dabei ist, dass Aussagen sich untereinander beeinflussen. Als einfaches Beispiel kann angenommen werden, dass eine Ontologie die beiden Aussagen „Ein Haus hat eine Tür“ und „Durch eine Tür kann man ein Objekt betreten“ enthält. Möchte man nun Wissen über Häuser aus der Ontologie extrahieren, wäre ein erster Ansatz, alle Aussagen, welche den Begriff „Haus“ enthalten, aufzunehmen. Dann bestände das Modul jedoch nur aus der ersten Aussage und man würde die Information, dass man ein Haus (durch die Tür) betreten kann, verlieren. Logikbasierte Module

bieten die Möglichkeit, zu einer Spezifikation, die beschreibt, welches Wissen im Modul enthalten sein soll, ein Modul zu bestimmen. Dieses Modul erfüllt dann bezogen auf die Spezifikation Garantien dafür, welches Wissen im Modul enthalten ist. Der Prozess der Modulextraktion kann also automatisiert durchgeführt werden, sofern eine ausreichende Beschreibung des Moduls vorliegt. Diese muss von Nutzenden erstellt werden.

In dieser Arbeit soll der Frage nachgegangen werden, wie der Prozess der Modulextraktion durch Visualisierungen unterstützt werden kann. Dabei wird auch betrachtet werden, wie sich der Prozess und die Visualisierungen bei großen Ontologien verhalten.

Bestehende Visualisierungen werden anhand von Kriterien verglichen, welche sich sowohl auf inhaltliche Anforderungen als auch auf Nutzbarkeitsaspekte beziehen. Einen grundlegenden Unterschied macht dabei, wie die Nutzenden das Modul spezifizieren. Die erste Möglichkeit ist, festzulegen, welches Vokabular der Ontologie von Interesse ist. Solche Verfahren werden hier signaturbasierte Verfahren genannt. Die zweite, in dieser Arbeit entwickelte Möglichkeit ist, Teile der Ontologie zu wählen, welche in dem Modul enthalten sein sollen. Dafür muss die Ontologie jedoch zuerst in Teile unterteilt werden. Diese Unterteilung wird auch Dekomopsition genannt und die entsprechenden Verfahren sind daher dekompositionsbasiert. Da sich signaturbasierte und dekompositionsbasierte Verfahren stark unterscheiden, werden sie separat betrachtet. Des Weiteren wird ein neues dekompositonsbasiertes Verfahren erarbeitet. Diese Arbeit ist eine Betrachtung theoretischer Grundlagen und wird daher keine Nutzerstudie enthalten.

Aufbau der Arbeit

In Kapitel 2 wird zunächst in die Themen Ontologien und Visualisierungen eingeführt. Anschließend beschreibt Kapitel 3 den Prozess der Modulextraktion und beschäftigt sich somit auch mit der Definition von Modulen.

Die Untersuchung der Visualisierungen findet in Kapitel 4 und 5 statt. Dabei beschäftigt sich Kapitel 4 mit Ansätzen, bei welchen die Nutzenden die Signatur als Spezifikation für das Modul wählen und Kapitel 5 mit Ansätzen, bei welchen die Ontologie in Teile unterteilt wird und aus diesen Teilen gewählt werden kann.

Nach den vergleichenden Kapiteln folgt in Kapitel 6.1 die Auswahl eines Ansatzes zur Umsetzung und Kapitel 6, welches sich mit der Umsetzung der aus dem Vergleich gewonnenen Erkenntnissen in einer Anwendung beschäftigt. Abschließend folgt der Abschluss der Arbeit in Form des Fazits in Kapitel 7.

Kapitel 2

Grundlagen

Im Folgenden werden Themengebiete, welche im Zusammenhang mit der bearbeiteten Fragestellung stehen, eingeführt. Diese lassen sich schon dem Titel entnehmen. Zum einen wird sich mit einem Prozess (Modulextraktion) im Zusammenhang mit Ontologien beschäftigt. Daher behandelt Abschnitt 2.1 Ontologien. Außerdem sollen Visualisierung zur Unterstützung dieses Prozesses untersucht werden. Dazu wird Informationsvisualisierung in Abschnitt 2.2 eingeführt.

2.1 Ontologien

Für die Bearbeitung der Fragestellung dieser Arbeit ist das Themengebiet der Ontologien zentral. Daher wird dieses nun eingeführt. Zunächst stellt sich die Frage, um was es sich bei einer Ontologie handelt. Nachdem dies in Abschnitt 2.1.1 definiert wurde, wird anschließend der Begriff des Semantic Web eingeführt. Danach folgt eine Einführung in Beschreibungslogik als Ontologiesprache. In dieser werden alle wichtigen Begriffe im Zusammenhang mit Ontologien, welche im Laufe der Arbeit benötigt werden, eingeführt. Darauf folgen die Betrachtung von der Ontologiesprache OWL, Beispielen für Ontologien und Anwendungen für Ontologien.

2.1.1 Defintion des Begriffes Ontologie

Der Begriff der Ontologie entstammt ursprünglich der Philosophie. In der Informatik kann eine Ontologie als eine explizite Spezifikation einer Konzeptualisierung aufgefasst werden. Diese Definition wurde in [Gru93] eingeführt und gilt laut [Stu11] als wohl meistzitierte Definition. Wie in [Stu11] beschrieben wird, ist für diese Definition zentral, was Konzeptualisierung und explizite Spezifikation in diesem Kontext bedeuten. In [BHS04] wird argumentiert, dass die Konzeptualisierung als abstraktes Modell für einen Teil der Welt zu verstehen ist und dieses Modell aus der Nennung von Eigenschaften wichtiger Konzepte und Beziehungen besteht. Als explizite Spezifikation sei zu verstehen, dass die Ontologie in einer eindeutigen Sprache definiert ist, welche sowohl von Menschen als

auch von Maschinen verstanden werden kann.

Um diese abstrakte Definition zu verdeutlichen, kann der Sachverhalt einer Universität betrachtet werden. In diesem wäre Professor ein Konzept. Als Eigenschaften eines Professors lassen sich aufführen, dass ein Professor ein Mensch ist, bei einer Universität angestellt ist und Vorlesungen hält. Dabei wären Mensch, Universität und Vorlesung jeweils wieder Konzepte. Eine Beziehung wäre hier das Angestelltenverhältnis eines Professors zu seiner Universität. Offensichtlich ist diese Beschreibung von Konzepten und Beziehungen im Kontext der Universität keine Ontologie. Dies lässt sich damit begründen, dass sie (im klassischen Sinne, ohne aufwendige Verarbeitung) nicht von einer Maschine verstanden werden kann.

2.1.2 Das Semantic Web

Ein Forschungsgebiet, welches großen Einfluss auf die Entwicklung von Technologien im Zusammenhang mit Ontologien gehabt hat, ist das Semantic Web. Dort spielen Ontologien eine zentrale Rolle für die Wissensrepräsentation (vgl.[BHS04]). Die Idee des Semantic Web ist, Inhalte im Internet nicht nur für Menschen zugänglich zu machen, sondern auch für Maschinen. Nutzt ein Mensch das Internet, um an Informationen zu gelangen, dann findet sie/er diese dort zumeist entweder als Text, Audiodatei oder Video. Die genannten Medien haben für den menschlichen Leser Vor- und Nachteile, sie sind jedoch für die meisten Menschen problemlos verständlich. Die Informationen aus diesen Medien automatisch zu extrahieren, um mit ihnen arbeiten zu können, ist für Maschinen jedoch nicht trivial. Wären Informationen im Internet in einer wohldefinierten Syntax und Semantik auch für Maschinen verständlich, dann würde dies verschiedene neue Möglichkeiten eröffnen, das Internet zu nutzen (vgl. [GC02a]). Ein solches Internet könnte als Semantic Web bezeichnet werden.

2.1.3 Beschreibungslogiken als Ontologiesprachen

Wie in 2.1.1 beschrieben, braucht eine Ontologie eine eindeutige Sprache, in welcher sie formuliert ist und welche sowohl von Menschen als auch von Maschinen verstanden werden kann. Als solche Sprache eignen sich Beschreibungslogiken besonders gut und werden in der Praxis auch eingesetzt (vgl. [BHS04]). Eine Alternative zu Beschreibungslogiken bilden beispielsweise F-Logiken (siehe [AL04]), welche in dieser Arbeit jedoch nicht betrachtet werden.

Mit Beschreibungslogiken lassen sich Wissensdatenbanken formulieren, welche als Ontologien verwendet werden können. In einer solchen Wissensdatenbank werden Konzepte und Rollen für eine Domäne formuliert. Dabei wird eine formale Semantik verwendet.

Durch die spezifischen Möglichkeiten, Aussagen zu treffen, unterscheiden sich die Beschreibungslogiken. Dabei werden verschiedene Operatoren zur Verfügung gestellt und auch die Kombinationsmöglichkeiten der Operatoren unterscheiden sich. Ein Vorteil von

Beschreibungslogiken ist, dass sich in ihnen automatisch schlussfolgern lässt. Im Generellen gilt, je ausdrucksstärker eine Beschreibungslogik ist, desto komplexer ist das Folgern. Die Untersuchung dieses Zusammenhanges und das Finden von guten Kompromissen zwischen Ausdrucksstärke und Komplexität ist eines der größten Forschungsfelder im Bereich der Beschreibungslogiken.

Im Folgenden werden Syntax und Semantik der Beschreibungslogik $\mathcal{ALC}$ eingeführt. Dies geschieht, um Begriffe im Kontext von Beschreibungslogiken formal einführen zu können. Für Ontologien verwendete Beschreibungslogiken wie $\mathcal{SHIQ}$(OWL) oder $\mathcal{SROIQ}$(OWL 2) unterscheiden sich von $\mathcal{ALC}$, das Verständnis der eingeführten Begrifflichkeiten hängt jedoch nicht von diesen Erweiterungen ab und wird im weiteren Verlaufe dieser Arbeit nicht benötigt. Daher genügt es hier $\mathcal{ALC}$ einzuführen. Die Definitionen stammen aus [Baa+17].

Eine der Haupteigenschaften von Ontologien ist zu konzeptionalisieren. Ein Konzept dient dem Zweck, für eine Menge an Individuen einen Teil derer Eigenschaften und Beziehungen zu repräsentieren. Atomare Konzepte werden durch Konzeptnamen beschrieben. Konzepte, welche man sich im Kontext von Bäumen vorstellen könnte, sind Baum, Laubbaum, Nadelbaum, Blatt, Stamm, usw..

Definition 2.1. Der Begriff **Konzept** stammt aus dem Bereich der Beschreibungslogik und beschreibt ein Prädikat, welches für jedes Element des Universums aussagt, ob dieses in der Erweiterung des Konzeptes ist. Konzepte werden im Kontext von Ontologien auch Klassen genannt. Dies ist eine Bezeichnung, welche vor allem aus dem Semantic Web bekannt ist.

Neben der Beschreibung von Konzepten ist Aufgabe einer Ontologie, Beziehungen zwischen Konzepten zu modellieren. Dafür dienen in der Beschreibungslogik die Rollen. Im Kontext von Bäumen könnte man sich Rollen vorstellen, welche die Beziehungen zwischen Teilen von Bäumen und den Bäumen modellieren. So könnte die Rolle „hängtAn“ die Blätter und Früchte, welche an diesem Baum hängen, mit dem Baum verbinden.

Definition 2.2. Der Begriff der **Rolle** stammt aus der Beschreibungslogik und beschreibt eine binäre Relation zwischen Individuen. Rollen werden im Kontext von Ontologien auch Eigenschaften genannt. Manchmal wird dabei zwischen „Data“ und „Object“ Properties unterschieden.

Nachdem die Bedeutung von Konzepten und Rollen in einer Wissensdatenbank bereits eingeführt wurde, soll definiert werden, wie ein solches Konzept aufgebaut sein kann.

Definition 2.3. Konzepte lassen sich in $\mathcal{ALC}$ wie folgt aufbauen :
Jeder Konzeptname ist ein Konzept. Seien C,D und E Konzepte und r eine Rolle, dann bilden die folgenden Ausdrücke Konzepte:

- $C \sqcup D$ (Disjunktion, gelesen C oder D)
- $C \sqcap D$ (Konjunktion, gelesen C und D)

- $\neg D$ (Negation, gelesen nicht C)
- $\exists r.D$ (Existenzbeschränkung)
- $\forall r.D$ (Wertbeschränkung)

Konzepte, welche nicht nur ein Konzeptname sind, werden auch zusammengesetzte Konzepte genannt. In Beschreibungslogik wird zwischen terminologischem Wissen und Objektwissen unterschieden. Dabei besteht das terminologische Wissen aus den Definitionen von Konzepten und Rollen, Objektwissen hingegen aus Wissen über Individuen. Das terminologische Wissen wird auch TBox genannt und das Objetkwissen ABox. Das Verhältnis zwischen terminologischem Wissen und Objektwissen kann sehr unterschiedlich sein. Es gibt Ontologien, welche kein Objektwissen enthalten, einige Ontologien enthalten jedoch auch größtenteils Objektwissen.

Definition 2.4. Seien A und B Konzepte, dann ist $C \sqsubseteq D$ eine **generelle Konzept Inklusion(GCI)**. Eine Menge solcher GCIs bildet die **TBox**, welche das terminologische Wissen enthält.

Eine mögliche Konzept Inklusion im Rahmen von Bäumen wäre $Apfelbaum \sqsubseteq Baum$, welche aussagt, dass jeder Apfelbaum ein Baum ist. Es dürfen selbstverständlich auch zusammengesetzte Konzepte verwendet werden.

Definition 2.5. Objektwissen besteht aus der Zuordnung von Individuennamen zu Konzepten und der Zurodnung von geordneten Paaren von Individuennamen zu Rollen. Sei I die Menge an Indivuennamen und $a, b \in I$. Zusätzlich sei C ein Konzeptname und und r ein Rollenname. Dann gilt:

- a:C ist eine Konzeptzuordnung, welche aussagt, dass das Individuum mit dem Namen a eine Instanz des Konzeptes C ist.
- (a,b):r ist eine Rollenzuordnung, welche aussagt, dass das Individuum mit den Namen a über die Rolle r mit dem Individuum mit dem Namen r in Beziehung steht.

Eine Menge von Zuordnungen wird Objektwissen oder ABox genannt.

Beispiel 2.6. Seien Apfel und Apfelbaum Konzeptnamen, hängtAn eine Rolle und a und b Individuennamen. Dann sind a:Apfel und b:Apfelbaum Konzeptzuordnungen und (a,b):hängtAn eine Rollenzuordnung. Diese würden zusammen ergeben, dass a ein Apfel ist, b ein Apfelbaum und a an b hängt.

Mit der Definition von terminologisches Wissen und Objektwissen kann definiert werden, was eine Wissensdatenbank ist.

Definition 2.7. Eine **Wissensdatenbank** $\mathcal{K}$ ist in der Beschreibungslogik als $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ definiert, wobei $\mathcal{T}$ das terminologische Wissen und $\mathcal{A}$ das Objektwissen beinhaltet.

Eine solche Wissensdatenbank kann nach der eingeführten Definition als Ontologie bezeichnet werden, da in einer eindeutigen Sprache durch Aufbau von Konzepten und

Rollen ein abstraktes Modell einer Welt erstellt wird. Wenn in dieser Arbeit von Ontologie die Rede ist, dann ist eine Ontologie gemeint, welche auf einer Wissensdatenbank, beschrieben in Beschreibungslogik, basiert.

Definition 2.8. Die einzelnen Aussagen, welche in TBox und ABox einer Ontologie in Form von GCIs, Konzept- und Rollenzuordnungen gemacht werden, werden **Axiome** genannt.

Um das Vokabular, welches von einer Wissensdatenbank verwendet wird, zu beschreiben, wird der Begriff der Signatur eingeführt. Dieser wird im späteren Verlauf dieser Arbeit im Zusammenhang mit Modulen eine zentrale Rolle spielen.

Definition 2.9. Das in einer Wissensdatenbank $\mathcal{K}$ verwendete Vokabular wird als **Signatur** Σ bezeichnet und besteht aus Konzeptnamen, Rollennamen und Individuennamen.

Der Begriff der Signatur wird in dieser Arbeit nicht nur für gesamte Wissensdatenbanken, sondern auch für einzelne Axiome und Mengen von Axiomen verwendet. Dabei wird für ein Axiom a die Signatur des Axioms als $\widetilde{a}$ notiert und die Signatur einer Menge von Axiomen M als $\widetilde{M}$.

Definition 2.10. Die **Domäne**, auch Universum genannt, definiert in der Beschreibungslogik den Bereich, über welchen Wissen repräsentiert werden soll. Dieser enthält eine Menge an Individuen, welche für den Bereich von Interesse sind.

Als Beispiel werden hier Bäume verwendet. Es soll Wissen über Bäume repräsentiert werden. Individuen gibt es in dieser Domäne sehr viele, da jeder existierende Baum ein solches darstellen würde.

Definition 2.11. Ein **Individuum** ist Bestandteil der Domäne und zeichnet sich durch Eigenschaften und mögliche Beziehungen zu anderen Indiviuuen aus.

Um die Bedeutung der einzelnen Operationen beim Konstruieren von Konzepten verstehen zu können, muss zunächst der Begriff der Interpretation eingeführt werden.

Definition 2.12. Eine **Interpretation** $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ besteht aus einer Menge an Individuen $\Delta^{\mathcal{I}}$ und einer Abbildung, welche

- jedem Konzeptnamen C eine Menge $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ zuweist.
- jeder Rolle R eine binäre Relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ zuweist.

Eine Interpretation ist also eine Zuordnung von Individuen(-paaren) zu Konzeptnamen und Rollennamen. Welche Individuen zusammengesetzten Konzepten in einer Interpretation zugeordnet werden, definiert die nächste Definition.

Definition 2.13. Seien C und D Konzepte, r eine Rolle und $\mathcal{I}$ eine Interpretation, dann gilt:

- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
- $(\exists r.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid$ es gibt ein $c \in \Delta^{\mathcal{I}}$ so, dass $(d,c) \in r^{\mathcal{I}}$ und $c \in C^{\mathcal{I}}\ \}$
- $(\exists r.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid$ für alle $c \in \Delta^{\mathcal{I}}$ gilt, wenn $(d,c) \in r^{\mathcal{I}}$ dann $c \in C^{\mathcal{I}}\ \}$

Nachdem nun die Syntax und Semantik für zusammengesetzte Konzepte eingeführt wurde, folgt hier ein Beispiel für die einzelnen Arten von Konstruktionen für jeweils ein Beispiel.

Beispiel 2.14. Seien Apfelbaum, Birnenbaum, GroßerBaum, Apfel und Blatt Konzeptnamen und hängtAn eine Rolle. Dann haben die folgenden Konzeptbeschreibungen die genannte Bedeutung.

- Apfelbaum ⊔ Birnenbaum: Alle Elemente, die Apfel- oder Birnenbaum sind
- Apfelbaum ⊓ GroßerBaum: Alle Apfelbäume, die auch große Bäume sind
- ¬Apfelbaum: Alle Elemente die kein Apfelbaum sind
- ∃hängtAn.Apfel: Alle Elemente, an welchen mindestens ein Apfel hängt
- ∀hängtAn.Blatt: Alle Elemente, an welchen ausschließlich Blätter hängen

Definition 2.15. Ein GCI wird von einer Interpretation $\mathcal{I}$ erfüllt, wenn gilt $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Intuitiv drückt man damit aus, dass jedes Individuum, welches Instanz des Konzepts A ist, automatisch auch Instanz des Konzepts B ist. Eine Interpretation erfüllt eine TBox, wenn alle enthaltenen Axiome erfüllt sind.

Definition 2.16. Damit eine Interpretation $\mathcal{I}$ eine ABox erfüllen kann, muss diese zunächst jedem Individuumnamen ein Element der Interpretation zuordnen. Sei a ein Name eines Individuums, dann wird das in der Interpretation dem Namen zugewiesene Element als $a^{\mathcal{I}}$ bezeichnet. Außerdem müssen alle Zuordnungen der ABox erfüllt werden:

- Eine Konzeptzuordnung a:C ist erfüllt, wenn gilt $a^{\mathcal{I}} \in C^{\mathcal{I}}$.
- Eine Rollenzuordnung (a,b):r ist erfüllt, wenn gilt $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$

Definition 2.17. Eine Interpretation erfüllt eine Wissensdatenbank $\mathcal{K}$, wenn sie sowohl $\mathcal{A}$ als auch $\mathcal{T}$ erfüllt.

Anhand von den in der Ontologie enthaltenen Axiomen und der zugrundeliegenden Beschreibungslogik ist es möglich zu schlussfolgern, welche Aussagen noch gelten müssen. Die folgende Definition gibt eine Reihe von möglichen Schlussfolgerungen an, welche auf einer Wissensdatenbank möglich sind.

Definition 2.18. Sei $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ eine $\mathcal{ALC}$ Wissensdatenbank, C,D Konzepte und b ein Individuenname, dann gilt

1. C ist erfüllbar bezüglich $\mathcal{T}$, wenn es ein Modell von $\mathcal{I}$ von $\mathcal{T}$ gibt, in welchem es ein $d \in \Delta^{\mathcal{I}}$ gibt, für welches gilt $d \in C^{\mathcal{I}}$;
2. C wird von D bezüglich $\mathcal{T}$ subsumiert, geschrieben $\mathcal{T} \models C \sqsubseteq D$, wenn für jedes

Modell $\mathcal{I}$ von $\mathcal{T}$ gilt $C^{\mathcal{I}} \sqsubseteq D^{\mathcal{I}}$;

3. C und D sind äquivalent bezüglich $\mathcal{T}$, geschrieben $\mathcal{T} \models C \equiv D$, wenn für alle Modelle $\mathcal{I}$ von $\mathcal{T}$ gilt $C^{\mathcal{I}} = D^{\mathcal{I}}$
4. $\mathcal{K}$ ist konsistent, wenn es ein Modell von $\mathcal{K}$ gibt;
5. b ist eine Instanz von C bezüglich $\mathcal{K}$, geschrieben $\mathcal{K} \models b : C$, wenn für alle Modelle $\mathcal{I}$ von $\mathcal{K}$ gilt $b^{\mathcal{I}} \in C^{\mathcal{I}}$

Aus diesen Möglichkeiten der Schlussfolgerung ergeben sich Schlussfolgerungsprobleme, indem man fragt, ob eine Aussage aus der Ontologie folgt.

Alle dieser Schlussfolgerungensprobleme sind im Zusammenhang mit der Arbeit an Ontologien von Interesse. So bedeutet eine inkonsistente Ontologie meist, dass bei der Erstellung ein Fehler unterlaufen ist, da es keine Menge von Individuen gibt, welche durch die Ontologie repräsentiert werden können. Ähnlich kann für unerfüllbare Konzepte argumentiert werden. Auch äquivalente Konzepte sind eine hilfreiche Information beim Arbeiten mit einer Ontologie, auch wenn diese nicht immer einen Fehler darstellen müssen. Durch das erschöpfende Berechnen aller Subsumptionen zwischen Konzepten lässt sich in einer Ontologie die abgeleitete Konzepthierarchie bestimmen. Zum automatischen Lösen dieser Schlussfolgerungsprobleme gibt es die sogenannten Reasoner. Beispiele für solche Reasoner und eine Untersuchung von deren Performance kann in [Pan05] gefunden werden.

2.1.4 OWL als Ontologiesprache

Damit Wissensdatenbanken in Beschreibungslogiken in der Praxis verwendet werden können, bedarf es einer einheitlichen Sprache, für welche Anwendungen existieren, mit denen sich Ontologien erstellen, ändern und zum Schlussfolgern nutzen lassen. Eine solche Sprache ist die Web Ontology Language 2(OWL 2). Diese wurde vom World Wide Web Consortium(W3C) spezifiziert und folgt auf die Web Ontology Language(OWL). Oft wird von OWL gesprochen, wenn eigentlich OWL 2 gemeint ist. Dies wird auch in dieser Arbeit geschehen. OWL 2 bietet unterschiedliche Profile an, denen unterschiedliche Beschreibungslogiken zugrunde liegen. Während in OWL 2 DL die gesamte Syntax von OWL 2 verwendet werden darf, sind OWL 2 EL, OWL 2 QL und OWL 2 RL jeweils auf eine Teilmenge der gesamten Syntax beschränkt, welche für einen bestimmten Einsatzzweck genügt und dabei Komplexitäten für Schlussfolgerungsprobleme aufweisen. (vgl. [Cue+08])

2.1.5 Anwendungen für Ontologien

Ontologien bieten eine abstrakte Möglichkeit, Wissen zu repräsentieren. Daher sind sie auf sehr unterschiedliche Problemstellung und Anwendungsfälle anwendbar. Mögliche Anwendungsfälle beinhalten dabei:

- Datenintegration - Wissen aus unterschiedlichen Quellen in einem Kontext zusammenführen und zugänglich machen: Ontologien bieten die Möglichkeit, Metadaten über unterschiedlichste Quellen einheitlich darzustellen. Außerdem können Ontologien bei der Datenintegration zur Vereinheitlichung von Vokabular genutzt werden.
- Suche - Repräsentation von Metadaten eines Elements, um eine bessere Suche zu ermöglichen: Ontologien ermöglichen eine semantische Suche in Daten.
- Gewinnung von neuem Wissen - Aus einer Menge an bekanntem Wissen neue Erkenntnisse ziehen

In der Medizin werden Ontologien eingesetzt. Solche Ontologien sind auf dem Portal BioPortal[1] zu finden. Auch in der Robotik werden Ontologien verwendet, um Wissen zu repräsentieren (siehe [Hai+13]).

2.1.6 Beispiel Ontologien

Da Ontologien in der Praxis gänzlich unterschiedlichen Zwecken dienen, unterscheidet sich ihr Aufbau deutlich voneinander. In dieser Arbeit werden für Beispiele zwei Ontologien verwendet, deren primärer Zweck es ist, als Beispiel für eine Ontologie zu dienen. Dies ist die Koala Ontologie[2] und die Pizza Ontologie[3]. Die Koala Ontologie ist eine sehr kleine Beispielontologie, welche sich inhaltlich mit Koalas, deren Lebensräumen und Uniabschlüssen beschäftigt. Auch die Ontologie Pizza ist lediglich als Beispiel für eine OWL-Ontologie gedacht und wird in den Tutorial zu Protege als Beispiel verwendet. Im Gegensatz zu Koala ist sie jedoch deutlich größer, im Vergleich mit medizinischen Ontologien jedoch immer noch klein. Inhaltlich beschäftigt sich die Ontologie mit dem Pizzen, aus welchen Bestandteilen diese aufgebaut sind und welche Sorten es gibt.

Ontologien der Medizin lassen sich, wie bereits erwähnt, auf BioPortal finden. Diese sind jedoch meist sehr groß. Bekannte Beispiele sind SNOMED CT[4] und NCIT[5].

2.2 Informationsvisualisierung

Informationsvisualisierung ist das Nutzen von computergestützter, interaktiver, visueller Repräsentationen von abstrakten Daten, um diese zu verstehen (vgl. [CMS99]). Als Ziel von Informationsvisualisierung ist das Aufdecken von Strukturen und Zusammenhängen in den Daten zu verstehen. Um dieses Ziel zu erreichen, muss entschieden werden, welche Aspekte der Daten visualisiert werden müssen und auf welche Weise dies geschehen soll. Dies wird als das Hauptproblem der Informationsvisualisierung betrachtet. Dabei ist der erste Teil des Problems ein sehr domänenabhängiges Problem. Welcher Teil der Daten

[1] *https://bioportal.bioontology.org/*
[2] *https://protege.stanford.edu/ontologies/koala.owl*
[3] *https://protege.stanford.edu/ontologies/pizza/pizza.owl*
[4] *https://bioportal.bioontology.org/ontologies/SNOMEDCT*
[5] *https://bioportal.bioontology.org/ontologies/NCIT*

visualisiert werden sollte, hängt allein davon ab, welche Informationen relevant sind, um die Daten zu verstehen. Dabei sollte jedoch beachtet werden, dass mehr Informationen eine Visualisierung nicht zwangsläufig verständlicher machen, sondern gegebenenfalls unübersichtlich. Für den zweiten Teil des Problems, die Frage, wie die Daten visualisiert werden sollen, gibt es eine Reihe an Metaphern, welche unabhängig von einem speziellen Fall betrachtet werden können. Im Kontext von Ontologien und dem Semantic Web sind vor allem Metaphern zum Repräsentieren von nicht numerischen und nicht räumlichen Daten gefragt, da der Großteil der Daten weder numerisch noch räumlich ist. Mögliche Metaphern sind Graphen, Bäume und Maps. [FSH02]

In [Pau19] wurden fünf Faktoren eingeführt, welche die Effektivität einer Visualisierung beeinflussen:

- **Bildschirmeigenschaften**. Die Eigenschaften des Bildschirms, auf welchem die Viesualisierung angezeigt wird, beeinflusst diese stark. Von ihnen hängt nicht nur die mögliche Menge an visualisierten Informationen ab, sondern auch die Wahl der besten Metapher und der Interaktionsmöglichkeiten.
- **Die Metapher**. Die Wahl einer angemessenen Metapher spielt eine wichtige Rolle. Eine Metapher sollte danach beurteilt werden, wie gut sie die Struktur der zu repräsentierenden Daten abbilden kann.
- **Interaktivität** bietet den Nutzenden die Möglichkeit, große Mengen an Daten zu überblicken und zu entscheiden, welcher Teil der Daten angezeigt werden soll. Mögliche Interaktionen beinhalten Auswählen, Filtern und Zoomen.
- **Vorverarbeitung der Daten** soll die Menge an Daten reduzieren. Dabei muss bestimmt werden, welche Daten irrelevant sind.
- Die **menschliche Wahrnehmung** soll von einer effektiven Visualisierung möglichst gut ausgenutzt werden. Wie der Mensch Informationen wahrnimmt, ist stark von deren Darstellung abhängig.

In dieser Arbeit werden Informationsvisualisierung von Bedeutung sein, wenn eine Ontologie präsentiert werden soll, aus welcher ein Modul extrahiert werden soll.

Kapitel 3

Einführung in den Prozess der Modulextraktion

Modulexraktion beschreibt den Prozess des Extrahierens eines Modules aus einer bestehenden Ontologie. Um einen Überblick über diesen Prozess zu liefern, wird in diesem Kapitel zunächst das Konzept von Modulen eingeführt. Anschließend wird der in dieser Arbeit zentrale Anwendungsfall der Modulextraktion eingeführt sowie weitere Anwendungsfälle genannt und abschließend verschiedene Modularten betrachtet.

3.1 Module

Angenommen ein Materialforscher hat eine Ontologie erstellt, welche einen guten Überblick über Eigenschaften verschiedener liefert, erstellt, dann enthält diese Ontologie verschiedene Teilgebiete. So enthält sie Wissen über Holz, Metall und Beton und könnte Aussagen über Witterungsbeständigkeit und Belastbarkeit machen. Ist nur ein Teil des Wissens benötigt, kann man statt der gesamten Ontologie auch ein Modul verwenden, welches zum Beispiel ausschließlich Aussagen über Witterungsbeständigkeit und Holz treffen kann. Dieser Abschnitt beschäftigt sich damit, wie sich Module definieren lassen und welche Eigenschaften Module haben können.

Konservative Erweiterungen und Moduldefinition

Damit logikbasierte Module definiert werden können, wird zunächst das Konzept der konservativen Erweiterungen definiert, um anschließend darauf basierend Module zu definieren. Im Folgenden sei $\mathcal{L}$ eine Beschreibungslogik, $\mathcal{M}$ und $\mathcal{T}$ $\mathcal{L}$-TBoxen und Σ eine Signatur.

Definition 3.1. $\mathcal{T}$ ist eine **deduktive Σ-konservative Erweiterung (Σ-dCE)** von $\mathcal{M}$ bzgl. $\mathcal{L}$, wenn für alle GCI Axiome a über $\mathcal{L}$ mit $\tilde{a} \subseteq \Sigma$ gilt, dass $\mathcal{M} \models a$ genau dann, wenn $\mathcal{T} \models a$.

Im Folgenden schreiben wir in diesem Fall $\mathcal{M} \equiv_{\Sigma}^{dCE} \mathcal{T}$. Zwei Interpretationen $\mathcal{I}$ und $\mathcal{J}$

überdecken sich bezüglich Σ, wenn für alle $X \in \Sigma$ gilt $\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$ und $X^{\mathcal{I}} = X^{\mathcal{J}}$. Dies schreiben wir als $\mathcal{I}|_\Sigma = \mathcal{J}|_\Sigma$.

Definition 3.2. $\mathcal{T}$ ist eine **Modell Σ-konservative Erweiterung (Σ-dCE)** von $\mathcal{M}$ bzgl. $\mathcal{L}$, wenn gilt $\{I|_\Sigma \mid I \models \mathcal{M}\} = \{I|_\Sigma \mid I \models \mathcal{T}\}$.

Im Folgenden schreiben wir in diesem Fall $\mathcal{M} \equiv_\Sigma^{mCE} \mathcal{T}$.

Definition 3.3. $\mathcal{M}$ ist ein **dCE-basiertes(mCE-basiertes) Modul** von $\mathcal{T}$ für Σ, wenn $\mathcal{M} \subseteq T$ und $\mathcal{T}$ eine Σ-dCE (Σ-mCE) von $\mathcal{M}$ bzgl. $\mathcal{L}$ ist.

Eigenschaften von Modulen

Unterschiedliche Arten von logikbasierten Modulen haben unterschiedliche Eigenschaften. Zwei wichtige Eigenschaften für Module werden im Folgenden eingeführt. Dabei sei $x \in \{dCE, mCE\}$, $\mathcal{L}$ eine Beschreibungslogik, $\mathcal{M}$ eine Teilmenge von $\mathcal{T}$ $\mathcal{L}$-TBoxen und Σ eine Signatur.

Abgeschlossen Ein x-Modul ist abgeschlossen, wenn gilt, dass alles Wissen, welches mit der Signatur des Moduls formuliert werden kann und aus der ursprünglichen Ontologie folgt, auch aus dem Modul folgt. Diese Eigenschaft ist zentral, um mit dem Wissen des Moduls weiterarbeiten zu können.

Definition 3.4. M ist ein abgeschlossen x-Modul, wenn gilt, $\mathcal{M} \subseteq T$ und $\mathcal{M} \equiv^x_{\Sigma \cup \tilde{M}} \mathcal{T}$.

Erschöpfend Ein Modul ist erschöpfend, wenn aus der Ontologie ohne das Modul kein Wissen über die Signatur des Moduls mehr folgt, welches keine Tautologie ist.

Definition 3.5. $\mathcal{M}$ ist ein erschöpfendes §-Modul, wenn gilt, $\mathcal{M} \subseteq T$ und $\mathcal{T} \backslash \mathcal{M} \equiv^x_{\Sigma \cup \tilde{M}} \emptyset$

3.2 Wiederverwendung von Wissen durch Modulextraktion

Die Idee hinter Ontologien ist, eine formalisierte Form von Wissensrepräsentation darzustellen, aus welcher durch logisches Schließen neues Wissen gewonnen werden kann. Einer Ontologie wird durch das Hinzufügen neuer Axiome Wissen hinzugefügt. (Angenommen das Axiom folgt nicht bereits aus der bisherigen Ontologie). Aus der Praxis ist bekannt, dass, um umfangreiche Themengebiete abzudecken, eine große Menge an Axiomen benötigt wird. Beispiel hierfür sind Ontologien, die in der Medizin eingesetzt werden. Das manuelle Hinzufügen neuer Axiome ist daher ein sehr zeitaufwändiger Vorgang, der möglichst umgangen werden sollte. Daher bietet es sich an, Wissen aus anderen Ontologien zu übernehmen. Offensichtlich könnte die gesamte Ontologie bzw. all ihre Axiome übernommen werden und somit ihr gesamtes Wissen importiert werden. Da die Größe von Ontologien jedoch bereits ohne das Übernehmen ganzer anderer Ontologien ein Problem

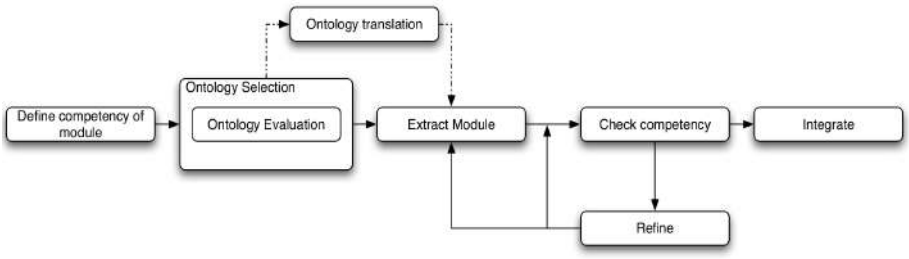

Abbildung 3.1 Prozess der Wiederverwendung von Wissen aus [DTI07]

bei der Arbeit mit diesen darstellt, sollte auch dies vermieden werden. Um das gesamte Wissen über ein Teilgebiet der Ontologie extrahieren zu können, werden Module verwendet.

Wie ein Arbeitsablauf zum Wiederverwenden von Wissen aussehen kann, wurde in [DTI07] untersucht. Als erster Schritt wird das Definieren der Kompetenz des zu extrahierenden Modules eingeführt. Anschließend folgt die Auswahl einer geeigneten Ontologie. Bei der dieser müssen die infrage kommenden Ontologien auf ihre Einsetzbarkeit überprüft werden. Dafür wird vorgeschlagen, die Ontologie sowohl auf die für das Modul festgelegte Kompetenz, als auch auf die Qualität der Ontologie zu überprüfen. Dabei sei unter anderem ihre Popularität ein Indiz. Nach der Auswahl der Ontologie folgt ein optionaler Schritt zur Übersetzung der Ontologie. Dieser ist vorgesehen, falls die verwendete Logik nicht mit dem Extraktionsprozess, welcher verwendet wird, kompatibel ist. Nachdem die bisherigen Schritte die Vorbereitungen getroffen haben, kann mit einem beliebigen Verfahren das Modul extrahiert werden. Hier könnte zum Beispiel eine Signatur gewählt werden, um ein $\bot$-Modul zu extrahieren. Nachdem ein Modul extrahiert wurde, solle man überprüfen, ob dieses Modul die im ersten Schritt definierten Anforderungen erfüllt. Ist dies nicht der Fall, kann einer oder mehrere Schritte des bisherigen Vorgangs angepasst und wiederholt werden. Wenn das richtige Modul gefunden wurde, kann dieses schlussendlich in die Ontologie integriert werden.

Das in dieser Arbeit stattfindende Evaluieren unterschiedlicher visueller Unterstützungen bei der Modulextraktion und das darauf basierend entworfene Tool berücksichtigt maßgeblich diesen Einsatzzweck. Präziser wird die Unterstützung des Schrittes der eigentlichen Extraktion des Moduls untersucht. Auch andere Schritte dieses Prozesses stellen Probleme da. So sind die in [DTI07] vorgeschlagenen Suchmaschinen für Ontologien inzwischen nicht mehr aktuell und auch das formale Definieren der Kompetenz eines Modules ist nicht trivial. Diese Probleme sollen jedoch nicht Inhalt dieser Arbeit sein.

3.3 Andere Anwendungsfälle

Neben der Wiederverwendung von Wissen sind Module noch für andere Anwendungsfälle nützlich. Diese werden in [PJC09] aufgeführt. Auf besonders großen Ontologien kann das Schlussfolgern teils sehr aufwendig und langsam sein. Daher kann es sich in diesem Fall lohnen, ein Modul zu extrahieren, welches das aktuell verwendete Wissen enthält und anschließend Anfragen an dieses Modul, anstatt an die gesamte Ontologie zu stellen (siehe z.B. [Kon+08]). Des Weiteren werden Module beispielsweise bei atomarer Dekomposition verwendet, um Ontologien strukturell zu zerlegen. Außerdem können Module verwendet werden, um Begründungen für bestimmte Folgerungen zu finden.

3.4 Modularten

Ein Modul wird üblicherweise für eine Signatur Σ, welche sich aus Konzeptnamen und Rollennamen der Ontologie zusammensetzt, extrahiert. Es ist ein schwieriges Problem, die in 3.1 eingeführten Module in ihrer minimalen Form zu extrahieren. In [GLW06] wurde für die Beschreibungslogik $\mathcal{ALC}$ gezeigt, dass das Problem, konservative Erweiterung zu entscheiden, 2EXPTime schwer ist. Daher werden in der Praxis Module verwendet, welche hinreichende Bedingungen für konservative Erweiterungen bieten, dabei jedoch größer als ein minimales Modul bezüglich konservativer Erweiterung sein können. Üblich sind dabei die in [Gra+08] eingeführten lokalitätsbasierten Verfahren. Feiner wird zwischen lokalitätsbaierten semantischen und syntaktischen Verfahren unterschieden, wobei semantische Lokalität mit Interpretationen der Ontologie arbeitet, während sich syntaktische Lokalität auf die Syntax der Axiome beschränkt. Semantische Lokalität lässt sich in $\emptyset$-Lokalität und Δ-Lokalität unterteilen. Folgende Definition aus [Gra+08] definiert, wann ein Axiom $\emptyset$-lokal ist.

Definition 3.6. Ein Axiom a über einer Logik $\mathcal{L}$ heißt $\emptyset$-lokal bzgl. Σ, wenn für jede Interpretation $\mathcal{I}$ eine Interpretation $\mathcal{J}$ existiert, sodass $\mathcal{I}|_\Sigma = \mathcal{J}|_\Sigma, \mathcal{J} \models a$ und für jedes $X \in \tilde{a}, X_\mathcal{J} = \emptyset$

Bei syntaktischer Lokalität hingegen wird zwischen $\top$-Lokalität und $\bot$-Lokalität unterschieden. Semantische Lokalität basiert auf einer Reihe an Bedingungen, welche hinreichend für semantische Lokalität sind. Es gilt, dass $\bot$-Lokalität $\emptyset$-Lokalität impliziert und $\top$-Lokalität Δ-Lokalität impliziert. In [SSZ09] wird argumentiert, dass die semantische Lokalität leichter berechnet werden kann, da hierfür kein Reasoner zu Hilfe genommen werden muss, sondern lediglich die Syntax der Axiome betrachtet wird. Das Schlussfolgern für ausdrucksstarke Beschreibungslogiken könne komplex werden. Syntaktische lokalitätsbasierte Module können verkleinert werden, indem die Extraktion von $\top$ und $\bot$ Modulen ineinander geschachtelt wird. Dies kann auch iterativ geschehen. Daraus

resultiert die Modulart der $\top\bot^*$-Module.

Mit einer beliebigen Art der Lokalität lässt sich ein Algorithmus für das Extrahieren eines Modules definieren. Dieser ist in Algorithmus 1 dargestellt.

Input : TBox T, Signatur Σ, $x \in \{\emptyset, \Delta, \bot, \top\}$
Output : x-Modul M aus O bzgl. Σ
$M \leftarrow \emptyset; T' \leftarrow T$ **repeat**
  $changed \leftarrow false$;
  for ***all*** $a \in T'$ **do**
    if *a nicht x-lokal bzgl.* $\Sigma \cup \tilde{M}$ **then**
      $M \leftarrow M \cup \{a\}$ $T' \leftarrow T'\{a\}$
until $changed = false$;
return M

Algorithmus 1 : Extraktion eines lokalitätsbasierten Modules aus [SSZ09]

Der Algorithmus arbeitet auf zwei Mengen von Axiomen. In T' werden zunächst alle Axiome der Ontologie gespeichert. M ist initial leer und wird am Schluss die Menge an Axiomen sein, welche das Modul definiert. In der ersten Iteration wird für jedes Axiom aus T' überprüft, ob es lokal in Bezug zur Signatur des gewünschten Modules vereinigt mit der momentanen Signatur von M ist. Ist dies nicht der Fall, wird dieses Axiom in M aufgenommen und aus T' gestrichen. Diese Iteration wird solange wiederholt, bis in einer Iteration kein neues Axiom aufgenommen wird und somit ein Fixpunkt erreicht ist. Es ist in [SSZ09] gezeigt worden, dass alle diese Module mCE-basierte Module sind und die in Kapitel 3.1 vorgestellten Eigenschaften erfüllen.

3.5 Differenzierung von signaturbasierten und dekompositionsbasierten Verfahren

In dieser Arbeit werden zwei unterschiedliche Arten von Verfahren der Modulextraktion betrachtet. Die erste Art sind die signaturbasierten Verfahren, welche eine Signatur als Eingabe benötigen. Die zweite Art sind dekompositionsbasierte Verfahren. Diese zerlegen die Ontologie zunächst in Teile und fordern dann als Eingabe eine Auswahl dieser Teile. Die zweite Art von Verfahren wird im Rahmen dieser Arbeit entwickelt.

Im Folgenden sollen die Eigenschaften der Arten gegenüber gestellt werden. Sowohl in den dekompositionsbasierten Verfahren als auch in den signaturbasierten Verfahren ist der Nutzer aufgefordert, durch seine Angaben zu spezifizieren, welches Wissen der Ontologie im Modul enthalten seien soll. Grundlegend unterscheiden sich die Verfahren jedoch durch die Art und Weise, wie diese Spezifikation stattfindet. Signaturbasierte Verfahren lassen den Nutzer eine Signatur bestimmen, welche man als „Interface“ des Moduls verstehen kann. Aus dieser kann durch Modulextraktionsverfahren der „Inhalt“ des Moduls bestimmt werden. Dekomposition hingegen bietet eine strukturierte Aufteilung des „In-

halts“ der Ontologie und ermöglicht den Nutzenden, gewünschte Teile des Inhalts in das Modul zu wählen. Das „Interface“ bzw. die Signatur ergibt sich automatisch.

3.6 Wichtige Informationen zu einem Modul

Die Nutzenden sollten ein Modul so präsentiert bekommen, dass sie einschätzen können, welchen Inhalt dieses hat. Um dies optimal umzusetzen, müssten die Axiome des Moduls angezeigt werden. Da diese oft umfangreich und für den Menschen schwer verständlich sind, sollte eine Alternative gefunden werden, welche bei guter Verständlichkeit und Übersichtlichkeit Informationen über das Modul repräsentiert.

Größe des Moduls Sei die Größe des Moduls die Anzahl der enthaltenden Axiome. Dann lässt sich aus der Größe des Moduls im Verhältnis zur gesamten Ontologie Wissen über das Modul erlangen. Bei Größe 0 steht sofort fest, dass das Modul kein Wissen enthält. Entspricht die Größe des Moduls der Größe der Ontologie, enthält das Modul die gesamte Ontologie. Im ersten Fall ist das Modul sinnlos, da es keinerlei Wissen enthält und im zweiten Fall muss den Nutzenden bewusst sein, dass das Modul äquivalent zur Ontologie ist und somit dessen Extraktion sinnlos ist. Führt nun eine Signatur Σ zu einem Modul $\mathcal{M}_1$ und die Erweiterung von Σ um ein Element α zu einem Modul $\mathcal{M}_2$, dann gilt, wenn die Größe von $\mathcal{M}_1$ gleich der Größe von $\mathcal{M}_2$ ist, fügt die Wahl von α Signatur kein neues Wissen zu dem Modul hinzu. Somit kann über die Größe des Moduls während dessen Wahl die ungefähre Menge an enthaltenem Wissen verfolgt werden.

Signatur des Moduls Ein anderer Faktor zur Einschätzung des Inhalts eines Moduls ist die Signatur des Moduls. Diese enthält das Vokabular, welches in den Axiomen des Moduls verwendet wird. Ist das Modul über eine Signatur spezifiziert worden (signaturbasierte Modulextraktion), wird diese auch Seed-Signatur genannt und kann mit der Signatur des Moduls verglichen werden. Dabei ist die Differenz der beiden Signaturen besonders interessant. Ist ein Element in der Seed-Signatur, jedoch nicht in der Modulsignatur, dann enthält das Modul kein Aussagen, welche dieses Element verwendet. Sind die Nutzenden an Aussagen interessiert, welche dieses Element verwenden, besteht die Möglichkeit die Seed-Signatur zu erweitern.

Ist ein Element in der Modulsignatur, jedoch nicht in der Seed-Signatur, dann ist dieses den Nutzenden möglicherweise noch unbekannt und bietet einen Ansatzpunkt für die weitere Exploration der Ontologie, da es eine Verbindung zu den in der Seed-Signatur gewählten Elementen gibt. Welche Form dieser Zusammenhang haben kann, hängt stark von der Modulart ab.

Kapitel 4

Vergleich signaturbasierter Verfahren

In diesem Kapitel sollen unterschiedliche Visualisierungen von Ontologien auf ihren Nutzen zur Wahl einer Signatur während der Extraktion eines Moduls aus einer Ontologie untersucht werden. Dazu wird zunächst die Frage nach sinnvollen Kriterien für diesen Vergleich bearbeitet, anschließend werden unterschiedliche Visualisierungsansätze vorgestellt und abschließend wird der Vergleich durchgeführt. Dabei soll ein Vergleich der theoretischen Möglichkeiten eines Ansatzes und kein Vergleich von implementierten Tools stattfinden, da Implementationsdetails einzelner Tools in dieser Arbeit nicht relevant sind. Tools werden als Beispiele für diese Möglichkeiten verwendet. Wenn hier von Konzepten und Rollen gesprochen wird, sind jeweils nur die atomaren Formen bzw. Konzept- und Rollennamen gemeint.

4.1 Kriterien des Vergleichs

Um unterschiedliche Ansätze einer Visualisierung zu vergleichen, werden in diesen Abschnitt Kriterien eingeführt, mit welchen sich ein Ansatz bezüglich des dargestellten Inhalts und bezüglich der Nutzbarkeit des Ansatzes bewerten lässt. Der Inhalt und die Nutzbarkeit einer Visualisierung können dabei als aufeinander angewiesen betrachtet werden, da eine Visualisierung sowohl Inhalt enthalten, als auch diesen verständlich darstellen muss, um ihn den Nutzenden zu vermitteln.

4.1.1 Inhaltliche Kriterien zur Visualisierung von Ontologien

In diesem Abschnitt werden eine Reihe an Kriterien eingeführt, welche in einem viel zitierten Überblickartikel([Kat+07]) als notwendig für die vollständige Repräsentation einer Ontologie erarbeitet wurden.

Darstellung der Konzepte Eine Visualisierung sollte in der Lage sein, alle Konzepte der Ontologie anzuzeigen. Dazu müssen bzw. sollen nicht alle Konzepte gleichzeitig angezeigt werden, es muss jedoch ermöglicht werden, zu jedem Konzept zu navigieren.

Darstellung der Individuen Die in der Ontologie enthaltenen Individuen sollten über die Visualisierung zugänglich sein. Der Zugang kann hierbei über ein Konzept, von welcher das Individuum eine Instanz ist, stattfinden. Es muss jedoch beachtet werden, dass es oft Konzepte mit vielen Individuen gibt und diese die Visualisierung leicht unübersichtlich machen. Des Weiteren ist ein Individuum oft Instanz vieler Konzepte.

Darstellungen der Konzepthierarchie Aus einer Visualisierung sollte für jedes Konzept deutlich werden, was die direkten Subkonzepte sind. Dies kann hilfreich sein, um einen Überblick über den Zusammenhang der verschiedenen Konzepte zu erlangen. Zusätzlich sollte an jeder Stelle, an welcher ein Konzept dargestellt wird, mindestens eines der Superkonzepte dieses Konzeptes dargestellt werden, sofern das Konzept solche hat. In der gesamten Visualisierung sollte es für jedes Konzept A, welches Superkonzept eines Konzeptes B ist, mindestens eine Darstellung dieser Beziehung durch ein Objekt, welches A repräsentiert, und eines für B geben.

Darstellung von Mehrfachvererbung Als besonderer Fall in der Konzepthierarchie kann es vorkommen, dass ein Konzept mehrere Superkonzepte hat. Dies sollte eine Visualisierung unterstützen, indem für alle Darstellungen eines Konzeptes alle Verbindungen zu übergeordneten Konzepten dargestellt werden.

Darstellung der Rollen Auch Rollen sollten in einer vollständigen Visualisierung repräsentiert sein. Dabei kann eine Rolle als Verbindung mit Beschriftung angezeigt werden.

Darstellung der Eigenschaften Eigenschaften von Objekten sollten in der Visualisierung zu finden sein.

4.1.2 Übertragbarkeit auf den Prozess der Extraktion

In diesem Abschnitt sollen die in Kapitel 4.1.1 vorgestellten Kriterien für eine vollständige Visualisierung einer Ontologie auf ihren Nutzen für die Modulextraktion untersucht werden.

Darstellung der Konzepte Konzepte spielen bei der Modulextraktion eine wichtige Rolle, da sie zur Signatur gehören, welche für das Modul gewählt wird. Deswegen ist es bedeutend, dass alle Konzepte repräsentiert sind und sich in die Signatur wählen lassen, damit die Nutzenden jedes Modul wählen können.

Darstellung der Individuen Individuen sind Teil der Signatur einer Ontologie und müssen daher bei der Extraktion von Nutzenden gewählt werden. Daher müssen sie in

einer guten Visualisierung zur Modulextraktion enthalten sein. In der Literatur wird die Existenz von Individuen bei der Extraktion von Modulen oft nicht betrachtet (siehe z.B. [Cue+09]), da diese den Prozess verkomplizieren und teilweise zu unschönen Effekten führen kann.

Darstellung der Konzepthierarchie Wie bereits dargestellt, ist das Wählen der Konzepte wichtig. Um einen Überblick über den Zusammenhang der Konzepte zu erhalten, ist die Konzepthierarchie eine gute Möglichkeit. Diese ist beim Erstellen der Ontologie relevant und kann außerdem durch Schlussfolgern zur inferierten Konzepthierarchie erweitert werden, welche mehr Informationen enthält.

Darstellung von Mehrfachvererbung Da es eine mögliche Strategie ist, zu einem gegebenen Konzept alle Superkonzepte wählen zu wollen, sollte die Mehrfachvererbung unterstützt werden.

Darstellung der Rollen Die Bedeutung für die Modulextraktion geht über die bisher eingeführte Version hinaus. Da die Möglichkeit, Rollen in die Signatur zu wählen, bestehen muss, ist neben ihrer Funktion als Verbindung zwischen Konzepten auch die Hierarchie der Rollen von Bedeutung.

Darstellung der Eigenschaften Auch Eigenschaften („Data Properties“) sind Bestandteil der Signatur und sollten somit in der Visualisierung repräsentiert sein.

Fazit Alle in Abschnitt 4.1.1 dargestellten Kriterien für die inhaltliche Vollständigkeit einer Repräsentation einer Ontologie sind auch im Kontext der Modulextraktion per Signatur von Bedeutung. Die Kriterien sind jedoch relativ stark auf Konzepte fokussiert und Rollen werden nur beiläufig betrachtet. Daher wird für diese Arbeit das Kriterium der Rollen um die Rollenhierarchie erweitert.

4.1.3 Kriterien der Nutzbarkeit

Eine Visualisierung sollte jedoch nicht ausschließlich anhand ihres Informationsgehaltes beurteilt werden, sondern auch anhand ihrer Nutzbarkeit. Um Visualisierungen von allgemeinen Daten im Zusammenhang mit dem Semantic Web abstrakt bewerten zu können, werden in [Fre+02] eine Reihe an Kriterien eingeführt. Diese sollen hier als Grundlage zur Bewertung der Nutzbarkeit der einzelnen Ansätze dienen. Da diese verwendet werden sollen um, statt einer spezifischen Umsetzung, generelle Ansätze zu bewerten, wurden Kriterien, welche hauptsächlich implementationsabhängig sind, vernachlässigt. Es folgt eine Auswahl an Kriterien aus [Fre+02], welche zur Untersuchung der Nutzbarkeit verwendet werden sollen.

Kognitive Komplexität Die kognitive Komplexität einer Visualisierung wird von der Menge der angezeigten Daten und deren Dimensionalität bestimmt. Eine kleine Menge an ähnlichen Daten ist für den Menschen leicht zu überblicken. Große Mengen an Daten, welche jedoch sehr ähnlich sind, und kleine Mengen an Daten, welche sich stark unterscheiden, sind auch noch zu überblicken. Wenn jedoch eine große Menge an Daten angezeigt wird und diese Daten sich in vielen Eigenschaften unterscheiden, sind dies nur schwer zu überblicken.

Räumliche Anordnung Die Räumliche Anordnung bewertet die Anordnung der Daten in der Visualisierung. Dabei kann betrachtet werden, wie einfach ein gesuchtes Element in der Visualisierung zu finden ist. Auch die generelle Verteilung der Daten sollte betrachtet werden. Mögliche Indizien sind die Verständlichkeit der Positionierung eines Elementes in der Ansicht und das Vorkommen von Verdecken von Daten durch andere.

Kodierung von Daten Dieses Kriterium betrachtet, wie ein Datenpunkt auf ein visuelles Element übertragen wird. Dabei sollte die Kodierung leicht verständlich sein. Außerdem kann die Verwendung von Symbolik untersucht werden.

Navigation Hier soll betrachtet werden, wie sich die Nutzenden in der Visualisierung bewegen können. Dazu zählt offensichtlich das Verschieben des Blickpunktes, Zoomen und das Verschieben von Elementen. Außerdem ist relevant, wie Elemente ausgewählt werden können.

Datensatz Reduktion Da große Mengen an Daten betrachtet werden sollen, ist es ungünstig, den gesamten Datensatz darzustellen. Daher sollten Techniken zur Reduktion der angezeigten Datenmenge betrachtet werden.

4.2 Auswahl

In diesem Abschnitt werden drei Ansätze zur Visualisierung von Ontologien vorgestellt. Für jeden dieser Ansätze werden Anwendungen untersucht, welche diese nutzen. Begonnen wird mit eingerückten Listen, anschließend werden Bäume betrachtet und abschließend Graphen.

4.2.1 Eingerückte Listen

Eine klassische Ansicht einer Ontologie sind die eingerückten Listen. Zu finden ist diese Ansicht beispielsweise im weit verbreiteten Open-Source-Ontologieeditor Protege[1]. Bekannt ist diese Visualisierung vor allem aus Dateimanagern. Verwendet werden kann

[1] *https://protege.stanford.edu/*

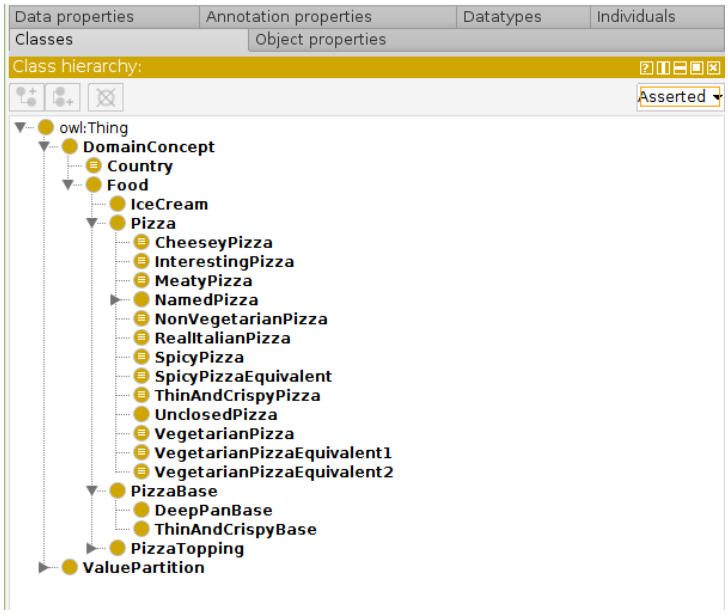

Abbildung 4.1 Konzepthierarchie in Protege für die Pizza-Ontologie

die Visualisierung für verschiedene Bestandteile der Ontologie. So kann die (abgeleitete) Konzepthierarchie als eingerückte Liste dargestellt werden. Ein Konzept steht genau dann eingerückt unter einem anderen Konzept, wenn diese ein Unterkonzept des anderen ist. Wenn gilt $A \sqsubseteq B$,$A \sqsubseteq C$ und $B \not\sqsubseteq C$,$C \not\sqsubseteq B$, dann sollte A sowohl unter B als auch unter C stehen. Auf dieselbe Weise kann auch die Rollenhierarchie abgebildet werden. Für mehr Übersichtlichkeit ist es oft möglich, die Kinder eines Elementes einzuklappen. Abbildung 4.1 zeigt ein Beispiel der Konzepthierarchie aus Protege.

4.2.2 Bäume

Eine andere Möglichkeit, Ontologien zu visualisieren, sind Bäume. Wie auch die eingerückten Listen lassen sich Bäume sowohl für die (abgeleitete) Konzept- als auch für die Rollenhierarchie einsetzen. Jedes Konzept(Rolle) entspricht dabei einem oder mehreren Knoten und eine Kante gibt an, dass ihr Ziel ein Unterkonzept ihres Ursprungs ist. Mehrmals ist ein Konzept (Rolle) auch hier repräsentiert, wenn sie Unterkonzept verschiedener Konzept (Rollen) ist, welche nicht auf einem Pfad liegen. Da ein Baum nur eine Wurzel haben kann, muss für jedes Konzept (Rolle), die kein Superkonzept hat, ein neuer Baum verwendet werden. Ein Blatt im Baum entspricht einer Konzept (Rolle), welche keine Unterkonzepte hat. Es gibt unterschiedliche Möglichkeiten, Bäume zu visualisieren. Eine Sammlung von Visualisierung bietet das Projekt Treevis.net[2]. Dieses führt zum Zeitpunkt der Erstellung dieser Arbeit über 300 Visualisierungsformen für Bäume auf. In der dem Projekt angegliederten Arbeit [Sch11] wird als zentrales Un-

[2] *https://treevis.net/*

terscheidungsmerkmal für Baumvisualisierungen die Kantenrepräsentation eingeführt. Dieses unterscheidet zwischen expliziten und impliziten Kanten. Eine explizite Kante ist selbst in der Visualisierung sichtbar(z.B klassischer Pfeil). Eine implizite Kante ist nicht sichtbar, wird jedoch durch die Positionierung der Knoten verdeutlicht. Beispielhaft sind hier TreeMaps, bei denen ein Knoten, welcher in einem anderen liegt, implizit dessen Kind ist. Die Bäume, welche, wie beschrieben, die Konzept- bzw. Rollenhierarchie repräsentieren, sind in ihrer Wahl einer Visualisierung frei. Im Folgenden werden drei Tools betrachtet, welche Bäume zur Visualisierung verwenden.

Bäume in Jambalaya [3] Das Protege-Plugin Jambalaya, welches in [Sto+02] eingeführt wurde, bietet unterschiedliche Visualisierungen für Ontologien. Darunter befinden sich zwei klassische Baumvisualisierungen. Zum einen gibt es einen Baum mit expliziten Kanten. Dieser ist in Abbildung A.2 beispielhaft für die Koala-Omtologie gezeigt. Diese Visualisierung ist nur für die Konzepthierarchie und nicht für die Rollenhierarchie verfügbar. Außerdem werden keine Individuen angezeigt. Die verschachtelte Treemap, welche auch Bestandteil von Jambalaya ist, enthält hingegen auch Individuen. Jedoch ist auch diese Visualisierung nicht für die Rollenhierarchie implementiert. Ein Beispiel für dieselbe Ontologie ist in Abbildung A.3 zu finden. Bei beiden Visualisierungen ist es möglich, Teilgraphen einzuklappen und auszuklappen, um die Ansicht zu vereinfachen. Außerdem kann in beiden Ansichten ein Element mit seinem Namen gesucht werden. In der Treemap lässt sich des Weiteren der Fokus der Ansicht durch Klicken auf ein Element verschieben.

OntoViz [4] Auch OntoViz verwendet einen Baum, um die Konzepthierarchie zu visualisieren. Dabei wird jedoch im Gegensatz zu Jambalaya nicht immer der gesamte Baum angezeigt, sondern, sofern ein Konzept ausgewählt wurde, nur ein Teil, welcher von dem Konzept, das aktuell fokussiert ist, abhängt. Um zu einem ersten ausgewählten Konzept zu gelangen, kann entweder zunächst der gesamte Baum angezeigt werden oder die links neben dem Baum angezeigte eingerückte Liste verwendet werden. Für ein ausgewähltes Konzept können die Nutzenden bestimmen, wie viel Kontext angezeigt werden soll. Dabei kann bestimmt werden, wie tief die Kinder und wie hoch die Eltern verfolgt werden sollen. In Abbildung A.8 sieht man ein Beispiel aus der Ontologie Pizza, bei welchem das Konzept Pizza fokussiert ist und jeweils zwei Ebenen von Kindern und Eltern angezeigt werden.

CropCircles Im Ontologieeditor Swoop[5], vorgestellt in [Kal+06], wurde in einer der letzten veröffentlichten Versionen eine Visualisierung der Konzepthierarchie mit dem

[3] *https://thechiselgroup.org/jambalaya/*
[4] *https://protegewiki.stanford.edu/wiki/OntoViz*
[5] *https://github.com/ronwalf/swoop*

Namen CropCircles hinzugefügt. Diese wird in [PWG05] beschrieben und arbeitet, wie Treemaps, mit impliziten Kanten zur Visualisierung eines Baumes. Jedes Konzept wird dabei anstelle eines Rechteckes durch einen Kreis repräsentiert. Befindet sich ein Kreis innerhalb eines anderen, ist der erste ein Unterkonzept des zweiten. Auffällig ist, dass die Kreise in Swoop keinerlei Beschriftung haben. Erst durch das Bewegen der Maus über den entsprechenden Kreis wird der Name des Konzeptes sowie dessen Tiefe in der Hierarchie und die Größe, bestimmt über die Anzahl der enthaltenden Konzepte, angezeigt. Wie schon bei OntoViz gibt es auch hier wieder links neben der Grafik eine eingerückte Liste, über welche sich Konzepte in der Grafik wählen lassen.

4.2.3 Graphen

Graphen sind eine sehr flexible Möglichkeit, Ontologien zu visualisieren. Da Bäume Graphen sind, ist dieser Ansatz mächtiger als Bäume. Grundlage von graphbasierten Visualisierungen bilden oft die zuvor beschriebenen Bäume der Konzept- bzw. Rollenhierarchie. Diese werden um weitere Kanten ergänzt, welche weitere Informationen bieten, aber die Struktur eines Baumes zerstören. Es lassen sich verschiedene Typen von Kanten definieren, welche unterschiedliche Informationen enthalten. So kann es Kanten für die Konzepthierarchie und für Rollen geben. Beispiele für Tools, in denen Graphen als Visualisierung genutzt werden, sind die Protege Plugins Jambalaya, VOWL und OntoGraf.

Graphen in Jambalaya Jambalaya verfügt neben den zwei Ansichten aus dem Abschnitt der Bäume über zwei weitere Ansichten, welche in die Kategorie der Graphen einzuordnen sind. Zum einen gibt es die „Nested View“. Diese kann als Erweiterung der Treemap verstanden werden, da in dieser Ansicht die Konzepthierarchie als Treemap dargestellt wird. Zusätzlich werden hier die Rechtecke der jeweiligen Konzepte mit einer expliziten farbigen Kante verbunden, wenn diese beiden Konzepte über eine Rolle verbunden sind. In Abbildung A.4 sieht man diese Ansicht für die Koala-Ontologie mit einigen bereits ausgeklappten Konzepten.

Die zweite Ansicht ist die „Query View“. Um diese zu nutzen, müssen die Nutzenden zunächst ein Konzept spezifizieren, nach welchem sie suchen. Für diese wird die sogenannte Nachbarschaft angezeigt. Die Nachbarschaft besteht aus allen Konzepten, welche über die Konzepthierarchie oder Rollen direkt mit dem gesuchten Konzept verbunden sind. Die Nutzenden können die so entstandene Abbildung erweitern, indem sie die Nachbarschaft einer der angezeigten Konzepte ausklappen. Außerdem lassen sich die angezeigten Kanten nach ihrem Typ einschränken. Wenn eine Art von Kante den Graphen unübersichtlich macht, kann diese versteckt werden. In Abbildung A.5 ist die „Query View“ für die Koala-Ontologie und den Suchterm „Animal“ zu sehen. Ist man mit dem Layout des Graphen unzufrieden, ist es möglich, Knoten per Hand zu verschieben.

ProtegeVOWL [6] Bei VOWL(Visual Notation for OWL Ontologies)[7] handelt es sich um eine Spezifikation zur Visualisierung von OWL-Ontologien. Diese legt eine einheitliche Definition der Symbole, welche für Bestandteile einer Ontologie Visualisierung verwendet werden, fest. Es gibt einige Tools, welche VOWL nutzen. Unter anderem gibt es ProtegeVOWL und WebVOWL. Hier soll es vor allem um ProtegeVOWL gehen, aber WebVOWL verhält sich ähnlich. Im in [LNB14] beschriebenen ProtegeVOWL wird eine Ontologie immer im gesamten visualisiert. Der Graph besteht aus genau einem Knoten für jedes Konzept und expliziten Kanten für Konzepthierarchie und Rollen. Den Nutzenden ist es möglich, mit der Visualisierung zu interagieren, indem sie einzelne Knoten(Konzept) mit der Maus verschieben. Sofern nicht deaktiviert, bewegen sich alle Knoten ständig, um das bestmögliche Layout zu schaffen. Dies gelingt jedoch nicht immer. Um zu der in Abbildung A.6 dargestellten Ansicht auf die Pizza-Ontologie zu gelangen, mussten einzelne Knoten mit der Maus so verschoben werden, dass sich ein geordnetes Bild ergab.

OntoGraf[8] Das Protege-Plugin OntoGraph kann als Nachfolger der „Query View“ aus Jambalaya gesehen werden. Tatsächlich verwendet das Projekt Teile aus Jambalaya wieder (Jambalaya ist inzwischen eingestellt). Links neben der Ansicht des Graphen befindet sich eine eingerückte Liste, in welcher ein Konzept gewählt werden kann, das dann dem Graphen hinzugefügt wird. Des Weiteren kann man wie in Jambalaya ausklappen und nach Konzepten suchen. In Abbildung A.7 ist ein Ausschnitt von Ontograf unter der Verwendung der Pizza-Ontologie zu sehen.

4.3 Vergleich

Nachdem sowohl Kriterien als auch Kandidaten für einen Vergleich der Qualität unterschiedlicher Ansätze einer Visualisierung zur Unterstützung signaturbasierter Modulextraktion eingeführt wurden, kann dieser Vergleich in diesem Abschnitt durchgeführt werden. Dafür wird mit dem Vergleich der Ansätze anhand der inhaltlichen Kriterien begonnen, anschließend mit den Kriterien der Nutzbarkeit fortgefahren, um abschließend zu einem Fazit zu gelangen.

4.3.1 Inhalt

Der inhaltliche Vergleich wird im Folgenden für jedes der in Abschnitt 4.1 eingeführten Kriterien und für jeden Ansatz durchgeführt. Eine Zusammenfassung findet sich in Tabelle 4.1.

[6] *http://vowl.visualdataweb.org/protegevowl.html*
[7] *http://vowl.visualdataweb.org/*
[8] *https://protegewiki.stanford.edu/wiki/OntoGraf*

→Kandidaten ↓ Kriterien	Eingerückte Liste	Bäume	Graph
Konzepte	Jeweils mindestens 1 mal vorhanden	Jeweils mindestens 1 mal vorhanden	Genau 1 mal vorhanden
Individuen	Als eigene Liste	Als Blätter	Als Knoten
Konzepthierarchie	Einrückung	Kanten	Kanten(speziell gekennzeichnet zur Unterscheidung)
Mehrfachvererbung	Nein - Kind kann bei beiden Eltern angezeigt werden	Nein - Kind kann bei beiden Eltern angezeigt werden	Ja
Rollen	Rollenhierarchie in eigener Ansicht	Rollenhierarchie in eigener Ansicht	Rollenhierarchie in eigener Ansicht und Verbindung von Konzepten über Rollen
Eigenschaften	Für aktuelle Auswahl neben der Visualisierung	Durch Visualisierung des Knotens und/oder Popup	Durch Visualisierung des Knotens und/oder Popup

Tabelle 4.1 Übersicht des Vergleiches anhand der inhaltlichen Kriterien

Konzepte In eingerückten Listen und Bäumen ist jedes Konzept jeweils mindestens einmal repräsentiert. Gegebenenfalls kann ein Konzept, wie bereits beschrieben, auch unter mehreren Konzepten als Kind stehen. Im Gegensatz dazu kann in einem Graphen jedes Konzept genau einmal repräsentiert werden.

Individuen In eingerückten Listen können Individuen als eigene Liste aufgeführt werden. Für Bäume und Graphen zeigt OntoGraf, wie Individuen als Blätter bzw. Knoten eingebunden werden können.

Konzepthierarchie In eingerückten Listen wird die Konzepthierarchie durch die Einrückung dargestellt. In Bäumen wird die Konzepthierarchie durch die Kanten des Baumes repräsentiert. Das gilt auch für Graphen, wobei hier nicht alle Kanten zur Konzepthierarchie gehören und Kanten daher durch Farbe, Form oder Beschriftung unterschieden werden müssen.

Mehrfachvererbung In Bäumen und eingerückten Listen kann Mehrfachvererbung

nicht umgesetzt werden. In eingerückten Listen kann ein Element nur direkt unter genau einem anderen Element stehen und in Bäumen darf ein Knoten nur genau einen Vorgänger haben. Graphen hingegen sind in der Lage Mehrfachvererbung abzubilden.

Rollen In jedem Ansatz kann die Rollenhierarchie äquivalent zur Konzepthierarchie in einer eigenen Ansicht dargestellt werden. Bei Graphen kommt jedoch noch hinzu, dass Rollen auch als zusätzliche Informationen über Zusammenhänge im Konzeptgraphen verwendet werden können.

Eigenschaften Eigenschaften von Elementen können in allen Visualisierungen angezeigt werden. Eingerückte Listen bestehen meist aus verhältnismäßig flachen Elementen, weil viele Elemente übereinander stehen sollen. Daher ist in den Elementen wenig Platz für Eigenschaften. Da eingerückte Listen aber hauptsächlich in die Höhe wachsen, ist neben der Liste noch Platz, welcher für eine detaillierte Präsentation des ausgewählten Elementes genutzt werden kann. Bäume und Graphen hingegen nutzen den gesamten Bildschirm. Hier können allerdings Popups verwendet werden, um ein Element detailliert anzuzeigen.

Fazit Zusammenfassend lässt sich sagen, dass zum einen der Inhalt einer Ontologie durch eingerückte Listen und Bäume ähnlich gut dargestellt werden kann und zum anderen Graphen in der Lage sind, mehr Inhalt zu präsentieren. Hierbei unterscheiden sich Graphen von eingerückten Listen und Bäumen vor allem durch die Fähigkeit, Beziehungen zwischen Konzepten durch Rollen auszudrücken. Somit wäre nach inhaltlichen Kriterien ein Graph der Ansatz der Wahl.

4.3.2 Nutzbarkeit

Der Vergleich der Nutzbarkeit findet in diesem Abschnitt anhand der Kriterien aus Kapitel 4.1.3 statt. Zusammengefasst wird der Vergleich in Tabelle 4.2.

	Eingerückte Listen	Bäume	Graphen
Kognitive Komplexität	Für jedes Element einen Eintrag, Einrückung implizite Darstellung der Hierarchie, 1-2 Dimensionen (Elemente alle auf einer Dimension verteilt, 2. Dimension für Einrückung)	Für jedes Element einen Knoten und Kanten für die Hierarchie, Verteilung der Daten in 2 Dimensionen (Höhe und Breite des Baumes)	Für jedes Element einen Knoten und Kanten für Beziehungen zwischen den Knoten, Dimensionalität ist von Art und Menge der verwendeten Beziehungen abhängig
Räumliche Anordnung	Untereinander, Einrückung entsprechend der Hierarchie	Viele unterschiedliche, gut umsetzbare Möglichkeiten	Für große Graphen schwieriges Problem
Kodierung von Daten	Listeneinträge: Elemente Einrückung: Hierarchie	Knoten: Elemente Kanten: Hierarchie	Knoten: Elemente Kanten: unterschiedliche Beziehungen
Navigation	Navigieren durch Scrollen durch die Liste, Auswahl durch Klick auf Eintrag, kein Verschieben und kein Zoomen	Navigieren mit der Maus, Auswahl durch Klick auf Element, Verschieben von Elementen und Zoomen möglich	Navigieren mit der Maus, Auswahl durch Klick auf Element, Verschieben von Elementen und Zoomen möglich
Datensatz Reduktion	Einklappen von Teilen der Liste	Einklappen von Teilbäumen	Einführen von Kriterien, wie beispielsweise der Nachbarschaft aus Jambalaya

Tabelle 4.2 Übersicht des Vergleiches anhand der Kriterien der Nutzbarkeit

Kognitive Komplexität In einer eingerückten Liste ist für jedes Element des repräsentierten Typs (Konzepte,Rollen,Individuen) ein Eintrag in der Liste vorhanden. Die Listeneinträge sind die einzigen Elemente dieser Visualisierung. Sie sind alle untereinander angeordnet (in einer Dimension). Für das Einrücken zum Darstellen der Hierarchie wird jedoch eine zweite Dimension verwendet. Somit sind die Daten in 1-2 Dimensionen dargestellt, je nachdem, ob man das Einrücken als Dimension begreift. In Bäumen gibt es für jedes Element des repräsentierten Typs (Konzepte,Rollen,Individuen) einen

Knoten. Zusätzlich gibt es Kanten, welche die Hierarchie repräsentieren. Jeder Knoten wird visualisiert, die Kanten hingegen können auch implizit sein. Ein Baum hat eine Höhe und eine Breite und erstreckt sich somit über zwei Dimensionen. Die Knoten in einem Graphen unterscheiden sich nicht von den Knoten eines Baumes. Jedoch können andere und verschiedene Arten von Kanten verwendet werden. Eine Kante setzt dabei immer noch die Knoten an ihren Enden in Beziehung, dies kann jedoch statt durch die Hierarchie auch über eine Rolle geschehen. Wie Jambalaya mit der „Nested View“ zeigt, ist es möglich, einige der Kanten implizit und andere explizit zu repräsentieren. Die Dimensionalität hängt nun zentral von der Art und Menge der verwendeten Beziehungen ab. Jedoch steigt die Dimensionalität verglichen mit Bäumen an , sofern die Hierarchie und mindestens eine weitere Beziehung zwischen Elementen verwendet wird. Es lässt sich zusammenfassend sagen, dass die kognitive Komplexität von eingerückten Listen über Bäume bis zu Graphen steigt.

Räumliche Anordnung Für eingerückte Listen gibt es eine fest definierte räumliche Anordnung. Ein Element steht eingerückt unter einem anderen, wenn es in der Hierarchie unter diesem eingeordnet ist. Alle Elemente stehen untereinander. Für einen Baum hingegen gibt es viele verschiedene Ansätze für eine räumliche Anordnung. Für einen beliebigen Graphen ist es jedoch ein Problem, eine übersichtliche Anordnung zu finden. Dies wurde auch in den Tools deutlich. Bis auf ProtegeVOWL hat keine Visualisierung den Graphen im gesamten angezeigt, wobei die „Nested View“das Problem umgangen hat, indem ein Baum für die Hierarchie verwendet wurde und in diesen weitere Kanten eingezeichnet wurden. ProtegeVOWL hingegen schaffte ohne Eingreifen der Nutzenden nicht, eine übersichtliche Anordnung zu präsentieren.

Kodierung von Daten In einer eingerückten Liste sind die Elemente in Listeneinträgen kodiert und die Einrückung repräsentiert die Hierarchie. In einem Baum stellen die Knoten Elemente und Kanten die Hierarchie dar. Auch Graphen verwenden Knoten für die Elemente. Kanten repräsentieren hier Beziehungen zwischen den Elementen.

Navigation In einer eingerückten Liste kann man sich für gewöhnlich durch Scrollen bewegen. Die Auswahl eines Elementes kann durch einen Klick auf dieses Element umgesetzt werden. Verschieben von Elementen und Zoomen ist hier nicht sinnvoll. In Bäumen und Graphen kann die Bewegung in der Ansicht durch Bewegen der Maus bei gedrückter Maustaste umgesetzt werden. Auch hier kann die Auswahl eines Elementes durch einen Klick auf dieses umgesetzt werden. Hier kann Zoomen und das Verschieben von Elementen verwendet werden, um den Nutzenden die Möglichkeit zu bieten, die Ansicht nach den jeweiligen Wünschen anzuordnen.

Datensatz Reduktion Für alle Ansätze gilt, dass es möglich ist, die Menge der aktuell angezeigten Daten zu reduzieren. Im Prinzip funktioniert dies für alle Ansätze ähnlich. Bei eingerückten Listen und Bäumen können jeweils alle Kinder eines Elements eingeklappt werden und in Graphen lassen sich Kriterien, wie die Nachbarschaft aus Jambalaya, definieren, welche ein- und ausklappen ermöglichen.

Fazit Relevante Unterschiede der Ansätze haben sich vor allem in den Kriterien „Kognitive Komplexität“ und „Räumliche Anordnung“ gezeigt. In Graphen müssen gegebenenfalls mehr Dimensionen der Daten angezeigt werden. Dies resultiert in dem Problem, dass eine gute räumliche Anordnung des gesamten Graphen kaum möglich ist. Bäume und eingerückte Listen hingegen haben ein einfacheres kognitives Modell und lassen sich besser im Raum anordnen. Daher wäre aus Gesichtspunkten der Nutzbarkeit eine dieser Ansätze wünschenswert.

4.3.3 Fazit

Die theoretische Betrachtung der Ansätze hat einen Konflikt zwischen Anforderungen an den Inhalt und an die Nutzbarkeit einer Visualisierung einer Ontologie dargelegt. Während Graphen die Möglichkeit besitzen, möglichst viel Inhalt zu repräsentieren, welcher für die Extraktion relevant ist, führt das Layout des Graphen zu Problemen. Eingerückte Listen und Bäume hingegen repräsentieren weniger Inhalt, können den Nutzenden jedoch übersichtlicher angezeigt werden. Unterstützt werden die Ergebnisse der Nutzbarkeitsanalayse von einer Studie, welche die Effektivität von verschiedenen Visualisierungen im Bezug auf die Unterstützung unterschiedlicher Aufgaben untersucht hat. Nutzende konnten die Aufgaben mit dem ClassBrowser, bei welchem es sich um eine eingerückte Liste handelt, signifikant schneller durchführen als mit dem Graphtool Jambalaya (siehe [Kat+06]). Diese Aufgaben waren zumeist darauf ausgelegt spezifische Informationen in der Ontologie zu finden. Für den Extraaktionsprozess ist jedoch ein Gesamtüberblick über die Ontologie vonnöten. Eine Lösung für den Konflikt zwischen Inhalt und Nutzbarkeit, welche einige der betrachteten Tools angewandt haben, ist die parallele Nutzung zwei verschiedener Visualisierungen. So kann eine eingerückte Liste, welche vor allem auch nicht den gesamten Bildschirm beansprucht, neben einem Graphen angezeigt werden. Eine solche Visualisierung erscheint zur Unterstützung der Wahl einer Signatur zur Modulextraktion wünschenswert.

4.4 Modifikation im Kontext der Modulextraktion

Wenn eine Visualisierung nun speziell zur Modulextraktion entwickelt wird, dann kann diese Visualisierung zusätzliche Informationen zum aktuellen Stand der Extraktion bieten. Bei der Wahl einer Signatur handelt es sich um einen stark iterativen Prozess. Die

Signatur wird Element für Element gewählt, wobei die Reihenfolge keine Rolle spielt. Unter Verwendung von Modularten, welche sich effizient berechnen lassen (z.B. syntaktische lokalitätsbasierte Module), ist es möglich, einen „Zwischenstand“ durch Berechnung des Moduls für die aktuelle Signatur zu bestimmen. Die Eigenschaften des so entstehenden Moduls können anschließend zum Visualisieren des aktuell gewählten Moduls verwendet werden. Dabei können alle Elemente der Signatur des Moduls in der Visualisierung markiert werden, um den Nutzenden zu zeigen, dass diese aktuell im Modul sind. Außerdem kann die Anzahl der enthaltenen Axiome als Maß für die Größe des Moduls angezeigt werden.

Kapitel 5

Dekompositionsbasierte Verfahren

In diesem Kapitel soll eine alternative Idee zur signaturbasierten Modulextraktion dargestellt werden, welche auf Möglichkeiten der Zerlegung von Ontologien beruht. Zunächst werden Kriterien aufgestellt, die ein Ansatz erfüllen sollte. Darauf folgend werden zwei Techniken der Dekomposition vorgestellt, welche für einen solchen Vorgang infrage kämen. Basierend auf der ausgewählten Technik wird ein Verfahren zu Modulextraktion beschrieben. Dieses wird anhand der eingeführten Kriterien bewertet. Danach wird die ausgewählte Technik für den Anwendungsfall der Modulextraktion erweitert und es wird ein Verfahren auf Grundlage der Erweiterung beschrieben sowie bewertet.

5.1 Dekomposition

Eine Dekomposition einer Ontologie ist die Zerlegung der Ontologie in verschiedene zusammenhängende Teile. Diese Teile der Dekomposition sollten nicht mit Modulen der Ontologie verwechselt werden, auch wenn sie, wie Module, eine Teilmenge der Axiome einer Ontologie sind. Es kann dabei Abhängigkeiten zwischen den einzelnen Teilen geben. Die Dekomposition einer Ontologie hat mehrere Anwendungsfälle. Zunächst kann sie genutzt werden, um unabhängig voneinander unterschiedliche Teile der Ontologie weiterzuentwickeln. Des Weiteren kann sie zur Gewinnung eines Überblicks über den strukturellen Zusammenhang dienen. Wie sich Dekomposition zur Extraktion von Modulen einsetzen lässt, soll Thema dieses Kapitels sein. Die gängigen Ansätze zur Dekomposition sind nicht für Modulextraktion konstruiert worden, daher muss zunächst untersucht werden, wie sie sich für diesen Anwendungsfall eignet und welche Modifikationen gegebenenfalls durchgeführt werden müssen.

Grafisch lassen sich Dekompositionen als gerichteter Graph darstellen. Dabei werden Teile als Knoten dargestellt und Abhängigkeiten als Kanten.

5.2 Kriterien

Die in Kapitel 4.1 eingeführten Kriterien lassen sich für eine als Graph dargestellte Dekomposition nicht anwenden, da die Darstellung der Ontologie über eine Dekomposition keinen Überblick über die Zusammenhänge des Vokabulars der Ontologie liefert, sondern vielmehr Abhängigkeiten zwischen Teilen der Ontologie darstellt. Somit werden im Folgenden neue Kriterien aufgestellt, welche es ermöglichen, einen Extraktionsverfahren basierend auf der Dekomposition einer Ontologie zu bewerten.

Nachvollziehbarkeit Bei der Modulextraktion muss nachvollziehbar sein, welchen Einfluss die Wahl eines neuen Teils der Ontologie auf das Modul hat. Wählen die Nutzenden ein Teil der Ontologie in sein Modul, werden gegebenenfalls andere Teile automatisch in das Modul aufgenommen. Die hier betrachteten Dekompositionsarten bestimmen diese anderen Teile anhand einer Abhängigkeitsrelation zwischen den Teilen. Damit die Nutzenden zu jedem Zeitpunkt des Prozesses einen Überblick haben, was sich bereits im Modul befindet, sollten automatische Wahlen visualisiert werden. Besser ist noch, wenn aus der Visualisierung offensichtlich hervorgeht, was gewählt werden wird. So könnte eine Abhängigkeitsrelation in einer Visualisierung durch gerichtete Kanten explizit dargestellt werden und die Nutzenden können durch Folgen dieser Kanten Abhängigkeiten erkennen.

Granularität Ein Extraktionsverfahren sollte es den Anwendenden ermöglichen, ein Modul zu wählen, welches bezüglich des von ihnen gewünschten Bereiches starke logische Garantien bietet und dabei so klein wie möglich ist. Damit dies erreichen werden kann, sollte gewährleistet sein, dass die Auswahl granular genug ist, um nicht ungewünschtes Wissen mitwählen zu müssen.

Verständlichkeit der Teile Für jeden Teil der Ontologie muss verständlich sein, welches Wissen durch diesen repräsentiert wird. Dies ist hilfreich, da Nutzende an bestimmtem Wissen aus der Ontologie, interessiert sind und das Modul genau so wählen möchten, dass dieses Wissen enthalten ist. Dies kann jedoch nicht funktionieren, wenn die Visualisierung eine Aufteilung der Ontologie in Teile anzeigt, bei denen die Nutzenden nicht nachvollziehen können, was diese enthalten.

Skalierbarkeit der Visualisierung Die Visualisierung sollte in der Lage sein, große Ontologien übersichtlich anzuzeigen.

Skalierbarkeit der Berechnung Die Berechnung, welche durchgeführt werden muss, um die Visualisierung anzuzeigen, sollte eine nicht exponentielle Komplexität aufweisen, damit auch große Ontologien visualisiert werden können.

5.3 Mögliche Ansätze

In diesem Abschnitt werden zwei Verfahren der Dekomposition vorgestellt. Dies geschieht vor allem unter Berücksichtigung der Eignung für den Prozess der Modulextraktion.

5.3.1 Partitionen basierend auf $\mathcal{E}$-Connections

Der Ansatz der $\mathcal{E}$-Connections (eingeführt in [Kut+04]) ist ursprünglich zur Kombination verschiedener Theorien in einer Ontologie eingeführt worden. In [Cue+06] wird gezeigt, wie sich mit diesem Ansatz Ontologien in nicht überlappende Teilgebiete aufteilen lassen und dabei Abhängigkeiten zwischen den Teilgebieten angegeben werden können. Dabei werden die Teile so gebildet, dass Wissen über ein Konzept in einem Teilgebiet enthalten ist. Dies gilt auch für einen Teil der Rollen. Der Rest der Rollen bestimmt eine Abhängigkeitsrelation zwischen den Teilgebieten, welche als Verbindungsrelationen bezeichnet werden. Die einzelnen Teile der Dekomposition selbst sind keine Module mit logischen Garantien. Für jeden Teil lässt sich ein solches Modul bestimmen, indem den Verbindungsrelationen erschöpfend gefolgt wird. Die entstehenden Module sind mCE-Module und abgeschlossen. Zusätzlich wird eine abgeschwächte Form von erschöpfend erfüllt. Für Ontologien, deren Logik nicht schwerer ist als $\mathcal{SHOIQ}$, lässt sich die feinste $\mathcal{E}$-Connection in polynomieller Zeit berechnen. Ein entsprechender Algorithmus wurde in [Cue+06] eingeführt. Problematisch verhalten sich $\mathcal{E}$-Connections bezüglich der Granularität. In [Del+19] gibt es mehrere Beispiele für Ontologien, deren $\mathcal{E}$-Connection aus nur einem einzigen Teil besteht. Dies ließe sich oft durch Vermeiden von „Top-Level“-Konzepten verändern, für bestehende Ontologien sind $\mathcal{E}$-Connections jedoch unbrauchbar. Des Weiteren zeigen die Beispiele, dass $\mathcal{E}$-Connections von Ontologien, welche zumindest aus mehr als einem Teil bestehen, trotzdem nicht sonderlich granular sind. Somit sind sie für die präzise Wahl eines Moduls ungeeignet, obwohl die Skalierbarkeit und die logischen Garantien geeignet wären.

5.3.2 Atomare Dekomposition

Die atomare Dekomposition (eingeführt in [Del+11b]) verwendet eine signaturbasierte Modulart, um zusammenhängende Teile und Abhängigkeiten zwischen diesen zu bestimmen. Die Teile werden hier Atome genannt und ein Atom ist eine maximale Teilmenge von Axiomen, welche ausschließlich gemeinsam in Modulen vorkommen. Eine Abhängigkeit eines Atoms von einem anderen besteht, wenn jedes Modul, welches das erste Atom enthält, auch das zweite Atom enthält. Die atomare Dekomposition kann als kreisfreier gerichteter Graph betrachtet werden, wobei jeder Knoten ein Atom repräsentiert und die Abhängigkeitsrelation durch Kanten repräsentiert wird.

Atomare Dekomposition benötigt eine signaturbasierte Modulnotation $x\text{-mod}(\Sigma, \mathcal{O})$, auf welcher diese basiert. Für diese werden in [Del+19] von der atomaren Dekomposition

folgende Eigenschaften gefordert:

(M0) x-mod$(\Sigma, \mathcal{O})$ ist ein abgeschlossenes R_Σ-Modul von $\mathcal{O}$.

(M1) x-mod$(\Sigma, \mathcal{O})$ ist eine eindeutig bestimmte Teilmenge von $\mathcal{O}$.

(M2) x-mod$(\Sigma, \mathcal{O})$ ist eine Teilmenge von $\mathcal{O}$.

(M3) Die Funktion x-mod$(*, *)$ ist monoton im ersten Argument.

(M4) Für alle R_Σ-Module $\mathcal{M}$ von $\mathcal{O}$ gilt $\mathcal{M} = x\text{-mod}(\tilde{M}, \mathcal{O})$.

(M5) x-mod$(\Sigma, \mathcal{O})$ ist ein erschöpfendes R_Σ-Modul von $\mathcal{O}$.

(M6) Für jede Tautologie $\alpha \in x\text{-mod}(\Sigma, \mathcal{O})$ gilt $\alpha \in x\text{-mod}(\Sigma \cap \tilde{\alpha}, \mathcal{O})$

Welche Modulnotation letztendlich verwendet wird, ist den Anwendern überlassen. Die in dieser Arbeit bereits vorgestellten lokalitätsbasierten Module sind dabei eine Option. Wie schon bei den $\mathcal{E}$-Connections kann ein Modul zu einem Atom(Teil) bestimmt werden, indem den Abhängigkeiten des Atoms gefolgt wird und jedes dabei erreichte Atom in das Modul aufgenommen wird. Die Menge an so erreichten Atomen wird auch Hauptideal genannt. Die Atome des Hauptideals eines Atoms bilden ein minimales Modul für das Atom. Die so entstehenden Module sind x-Module (siehe [Del+19]). Ungünstigerweise lässt sich auf diese Weise nicht jedes x-Modul einer Ontologie extrahieren, da es Module gibt, welche zwei oder mehr Atome enthalten, die weder direkt noch indirekt über die Abhängigkeitsrelation in Verbindung stehen (siehe [Del+19]). Dieses Problem wird im weiteren Verlauf noch weiter ausgeführt. Eine Menge von Atomen, die, wie beschrieben, nicht in Verbindung stehen, wird auch Antikette genannt. Die Eigenschaften der so erhaltenden Module sind abhängig von der Modulart x. Daher erfüllen die durch die atomare Dekomposition bestimmten Module mindestens die Eigenschaften, welche sie von Modularten fordert, alle weiteren Eigenschaften der Modulart bleiben jedoch auch erhalten.

Auch die Berechenbarkeit der atomaren Dekomposition hängt stark von der Wahl der Modulart ab.Wie in [Del+11b] gezeigt wurde, muss in Abhängigkeit von der Größe der Ontologie linear oft ein x-Modul berechnet werden und anschließend in quadratischer Laufzeit die Abhängigkeitsrelation bestimmt werden. Wird bedacht, dass syntaktische lokalitätsbasierte Module verwendet und diese in polynomieller Zeit berechnet werden können, ergibt sich, dass die atomare Dekomposition durchaus in polynomieller Laufzeit berechnet werden kann.

5.4 Extraktion mit atomarer Dekomposition

Aus der Visualisierung des Graphen der atomaren Dekomposition einer Ontologie lassen sich theoretisch alle Module der der Dekomposition zugrundeliegenden Modulart wählen. In [Del+19] ist gezeigt, dass jedes Modul der Modulart eine Vereinigung von Hauptidea-

len bezüglich der Abhängigkeitsrelation von Atomen ist. Als Problem ist anzusehen, dass die Umkehrung nicht gilt. Nicht jede Vereinigung von Hauptidealen von Atomen ist ein Modul. Folglich können somit nur die in [Del+19] als „Genuine Modules“ eingeführten Module zuverlässig aus einer statischen Visualisierung der atomaren Dekomposition gewählt werden. Bei der Wahl von zwei oder mehr Atomen, welche nicht miteinander in Beziehung stehen, folgt aus der Abhängigkeitsrelation der atomaren Dekomposition nicht, welche Atome im resultierenden Modul enthalten sein werden.

Um dieses Problem zu umgehen, ist im Zuge dieser Arbeit ein dynamisches Vorgehen entwickelt worden, welches den Nutzenden Informationen bereitstellt, auch Module, welche auf größeren Antiketten basieren, informiert wählen zu können. Dieser ist in Algorithmus 2 dargestellt. Zunächst wird die atomare Dekomposition der Ontologie auf Basis einer signaturbasierten Extraktionsmethode bestimmt. Anschließend wird eine Menge für Atome, welche von Nutzenden gewählt werden, und eine Menge für das Modul selbst initialisiert. Danach wird die atomare Dekomposition als Graph visualisiert. Darauf folgt der iterative Prozess der Wahl der Atome. Markieren Nutzende ein Atom, werden alle Atome markiert, welche durch die Wahl dieses Atoms zusätzlich in das Modul gelangen würden. Sind sich Nutzende sicher, ein markiertes Atom aufnehmen zu wollen, können sie dies bestätigen und anschließend die Wahl von Atomen fortsetzen. Der beschriebene Algorithumus könnte in einer Umsetzung um weitere Optimierungen für eine optimale Übersicht über den Prozess erweitert werden. So ist denkbar, Atome in unterschiedlichen Farben zu wählen. Eine Farbe für explizit gewählte Atome, eine Farbe für implizit gewählte Atome und eine dritte für die Erweiterung durch die Wahl eines zusätzlichen Atoms.

Input : Eine Ontologie O, eine Funktion x-Modul(*,*) zur Berechnung von Modulen bezüglich einer Signatur
Output : x-Modul M aus O
Graph G = berechne atomare Dekomposition basierend auf Modulart x.;
$A = \emptyset$;
$M = \emptyset$;
repeat
  Visualisiere G;
  Nutzer wählt Atom a in G;
  M' =x-Modul($\tilde{A} \cup \tilde{a}$));
  Markiere alle Atome die in M' enthalten sind ;
  if *Nutzer bestätigt Wahl von a* **then**
    $A = A \cup a$ $M = M'$
until *Nutzer beendet Wahl von Axiomen*;
return M

Algorithmus 2 : Extraktion eines Modules basierend auf atomarer Dekomposition

5.4.1 Bezeichnung von Atomen

Um die atomare Dekomposition sinnvoll verwenden zu können, wird eine Möglichkeit benötigt, ein Atom zu beschriften. In [Del+11a] wird dieses Problem ausgeführt. In dieser Referenz wird ein Ansatz zur Beschreibung aller Signaturen, in denen ein Atom enthalten ist, unter dem Namen „Minimal Seed Signatures“ betrachtet. Die Berechnung und Darstellung dieser Beschreibungen führt jedoch zu Problemen. Für diese Arbeit wird eine Beschriftung eingeführt, welche auf der Signatur des Moduls, welches aus dem Atom resultiert, basiert. Diese wird Signaturerweiterung genannt.

Definition 5.1. Die Signaturerweiterung Σ für ein Atom $\mathfrak{a}$ sei die Signatur $\Sigma = \widetilde{\mathcal{M}_\mathfrak{a}} \setminus (\bigcup_{\mathfrak{b}|\mathfrak{b}\prec\mathfrak{a}} \widetilde{\mathcal{M}_\mathfrak{b}})$

Somit ist die Signaturerweiterung selbst eine Signatur und enthält, informell ausgedrückt, die in dem zugehörigen Atom zum ersten Mal vorkommenden Signaturelemente. Zur Signaturerweiterung sei gesagt, dass eine Signaturerweiterung eines Atoms leer sein kann, wenn das entsprechende Atom kein neues Vokabular einführt. Außerdem kann in einer atomaren Dekomposition ein Element in Signaturerweiterung von mehreren Atomen vorkommen.

5.4.2 Bewertung der Modulextraktion mit atomarer Dekomposition

Nachvollziehbarkeit Bei der Wahl des ersten Atoms in der atomaren Dekomposition ist ersichtlich, welche anderen Atome im Modul aufgenommen werden. Dazu muss den Abhängigkeiten gefolgt werden. Ab der Wahl eines weiteren Atoms kann es unübersichtlicher werden. Da in diesem Fall auch Atome im Modul enthalten sein können, welche von keinem der gewählten Atome abhängig sind, könnten im gesamten Graphen der Dekomposition an beliebiger Stelle Atome markiert sein. Dies zu überprüfen, kann bei großen Graphen für Nutzende umständlich sein.

Granularität Die Granularität der atomaren Dekomposition hängt von der gewählten Modulart ab. Die Evaluation in [Del+19] zeigt, dass die atomare Dekompositionen auf Basis von lokalitätsbasierten Modulen fein granulär ist.

Verständlichkeit der Teile Die hier verwendete Signaturerweiterung ist in der Lage, darzustellen, wie ein Atom die Signatur eines Moduls erweitert. Sie hat jedoch die bereits aufgeführten Nachteile.

Skalierbarkeit der Visualisierung Wie in der Einführung der atomaren Dekomposition bereits dargestellt, ist der Graph der atomaren Dekomposition recht granular. Daraus folgt, dass für Ontologien mit vielen Axiomen viele Atome vorhanden sind. Dadurch kann der Graph unübersichtlich werden. Im Gegensatz zu signaturbasierten Ver-

fahren gibt es hier noch keine Ansätze, wie nur der momentan geforderte Ausschnitt einer Dekomposition betrachtet werden kann.

Skalierbarkeit der Berechnung Für dieses Verfahren muss zunächst die atomare Dekomposition der Ontologie berechnet werden. Wie in der Einführung der atomaren Dekomposition dargestellt, muss dafür linear oft ein Modul berechnet werden. Außerdem muss hier bei jeder Wahl zusätzlich noch ein Modul berechnet werden. Zum einen lässt sich somit sagen, dass eine starke Abhängigkeit von der Modulart besteht, zum anderen ist zu erkennen, dass initial mit einer längeren Berechnung zu rechnen ist und während der Wahl der Atome kürzere Berechnungen durchgeführt werden müssen.

Ein offensichtlicher Vorteil dieses Verfahrens ist die Einfachheit. Im Gegensatz zu signaturbasierten Verfahren, bei denen Konzepte, Rollen und gegebenenfalls auch Instanzen gewählt werden müssen, genügt bei atomarer Dekomposition die Wahl von Atomen. Auch die Beziehung zwischen den Atomen kann durch die Abhängigkeitsrelation relativ einfach und aussagekräftig dargestellt werden. Da es in der Abhängigkeitsrelation keine Kreise geben kann, weil die Atome eines Kreises immer zu einem Atom zusammenfallen würden, lässt sich die atomare Dekomposition als kreisfreier gerichteter Graph darstellen. Man könnte von einem Baum mit Mehrfachvererbung sprechen. Aussagekräftig ist die Abhängigkeitsrelation bei der Modulextraktion dadurch, dass sie darstellt, welche anderen Teile (Atome) der Ontologie zwingend auch in einem Modul enthalten sind, sobald man ein Teil (Atom) wählt. Die Visualisierung von atomarer Dekomposition hat aktuell zwei zentrale Nachteile. Zunächst führt die hohe Granularität der Unterteilung dazu, dass die Dekomposition großer Ontologien leicht unübersichtlich wird. Außerdem ist die Abhängigkeitsrelation für die Wahl mehrerer Atome nicht geeignet.

5.5 Erweiterte atomare Dekomposition

Um eine bessere iterative Wahl von Modulen in der atomaren Dekomposition zu ermöglichen, kann es sinnvoll sein, die atomare Dekomposition nach der Wahl eines Atoms in das Modul umzustrukturieren. Dies kann das Problem des in Kapitel 5.4 beschriebenen Ansatzes, dass es nach der Wahl des ersten Atoms keine strukturelle Implikation über die mitgewählten Atome bei der Wahl eines zweiten Atoms mehr gibt, lösen. Hier wird ein Ansatz dafür entwickelt, welcher eine Erweiterung der atomaren Dekomposition darstellt und daher erweiterte atomare Dekomposition genannt wird.

Die grundlegende Idee der erweiterten atomaren Dekomposition ist, zur Berechnung der Atome nicht alle Module zu betrachten, sondern nur die Module, welche eine Menge $\mathcal{A}$ an Axiomen enthält. Dafür müssen die Begriffe der atomaren Dekomposition für die erweiterte atomare Dekomposition neu definiert werden.

Zunächst definieren wir formal die Einschränkung der Module auf die Module, welche

$\mathcal{A}$ enthalten.

Definition 5.2. Sei $\mathcal{O}$ eine Ontologie, $\mathcal{A} \subseteq \mathcal{O}$ und $\mathcal{M} \in \mathfrak{F}(\mathcal{O})$, dann ist $\mathcal{M}$ ein $\mathcal{A}$-Modul genau dann, wenn gilt $\mathcal{A} \subseteq \mathcal{M}$. Wir schreiben dann $\mathcal{M}^{\mathcal{A}}$.

Anschließend wird die Äquivalenzrelation, welche die Axiome in Atome aufteilt und der Begriff der Atome selbst eingeführt.

Definition 5.3. Sei $\mathcal{O}$ eine Ontologie, $\mathcal{A} \subseteq \mathcal{O}$ und $\alpha, \beta \in \mathcal{O} \setminus \mathcal{A}$. Dann gilt $\alpha \sim_{\mathcal{O}}^{\mathcal{A}} \beta$, wenn für alle $\mathcal{M}^{\mathcal{A}}$ gilt $\alpha \in \mathcal{M}^{\mathcal{A}}$ genau dann, wenn $\beta \in \mathcal{M}^{\mathcal{A}}$.

Definition 5.4. Die $\mathcal{A}$-Atome einer Ontologie $\mathcal{O}$ sind die Äquivalenzklassen von $\sim_{\mathcal{O}}^{\mathcal{A}}$. Die Menge der $\mathcal{A}$-Atome von $\mathcal{O}$ wird als $\mathfrak{A}_{\mathcal{A}}(\mathcal{O})$ bezeichnet.

Beobachtung 5.5. Für jede Ontologie $\mathcal{O}$ und jedes $\mathcal{A} \subseteq \mathcal{O}$ gilt:

1. $\mathfrak{A}_{\mathcal{A}}(\mathcal{O})$ ist eine Partitionierung von $\mathcal{O} \setminus \mathcal{A}$.
2. $\#\mathfrak{A}_{\mathcal{A}}(\mathcal{O}) \leq \#(\mathcal{O} \setminus \mathcal{A})$
3. Jedes $\mathcal{A}$-Modul ist eine Vereinigung von $\mathcal{A}$-Atomen vereinigt mit $\mathcal{A}$.

Nachdem Atome eingeführt sind, wird die Verbindung der Atome in Form der Abhängigkeitsrelation benötigt.

Definition 5.6. Seien $\mathfrak{a}$,$\mathfrak{b}$ $\mathcal{A}$-Atome einer Ontologie $\mathcal{O}$. Wir sagen $\mathfrak{a}$ ist abhängig von $\mathfrak{b}$ (geschrieben $\mathfrak{a} \succeq_{\mathcal{A}} \mathfrak{b}$ bzw. $\mathfrak{b} \preceq_{\mathcal{A}} \mathfrak{a}$), wenn für alle $\mathcal{A}$-Module $\mathcal{M}^{\mathcal{A}}$ gilt $\mathfrak{a} \in \mathcal{M}^{\mathcal{A}}$ impliziert $\mathfrak{b} \in \mathcal{M}^{\mathcal{A}}$. Die Relation $\succeq_{\mathcal{A}}$ wird Abhängigkeitsrelation genannt.

Mit diesen Bestandteilen lässt sich nun die erweiterte atomare Dekomposition(eAD) definieren.

Definition 5.7. Das Tupel $(\mathfrak{A}_{\mathcal{A}}(\mathcal{O}), \succ_{\mathcal{A}})$ wird erweiterte atomare Dekomposition(eAD) von $\mathcal{O}$ bezüglich $\mathcal{A}$ genannt.

Das folgende Lemma und Korollar werden in dieser Arbeit vor allem benötigt, um zu argumentieren, warum der vorgestellte Algorithmus zur Berechnung der erweiterten atomaren Dekomposition korrekt ist.

Lemma 5.8. Für jedes $\mathcal{A}$-Atom $\mathfrak{a} \in \mathfrak{A}_{\mathcal{A}}(\mathcal{O})$ und jedes Axiom $\alpha \in \mathfrak{a}$ gilt, dass $\mathcal{M}_{\alpha}^{\mathcal{A}}$ das kleinste $\mathcal{A}$-Modul von $\mathcal{O}$ ist, welches $\mathfrak{a}$ enthält.
Beweis.

1. $\mathfrak{a} \in \mathcal{M}_{\alpha}^{\mathcal{A}}$
   Sei $\mathfrak{a} \in \mathfrak{A}_{\mathcal{A}}(\mathcal{O})$ und $\alpha \in \mathfrak{a}$. Wegen Exhausivität gilt $\mathcal{O} \setminus \mathcal{M}_{\alpha}^{\mathcal{A}} \equiv_{\tilde{\alpha}} \emptyset$. Da auch gilt $\mathcal{O} \models \alpha$, muss, sofern α keine Tautologie ist, $\alpha \in \mathcal{M}_{\alpha}^{\mathcal{A}}$ offensichtlich gelten. Ist α eine Tautologie, kann angenommen werden, dass es ein Modul $\mathcal{M} = mod(\Sigma, \mathcal{O})$ gibt, welches α enthält. Nach Eigenschaft (M6) gilt auch $\alpha \in mod(\Sigma \cap \tilde{\alpha}, \mathcal{O})$. Wegen Monotonie (M3) gilt folglich auch $\alpha \in \mathcal{M}_{\alpha}^{\mathcal{A}}$.
2. $\mathcal{M}_{\alpha}^{\mathcal{A}}$ ist das kleinste $\mathcal{A}$-Modul von $\mathcal{O}$, welches $\mathfrak{a}$ enthält.
   Sei $\mathcal{M}^{\mathcal{A}}$ ein Modul und gelte $\mathfrak{a} \in \mathcal{M}^{\mathcal{A}}$. Dann gilt:

$$
\begin{aligned}
\mathcal{M}_\alpha^{\mathcal{A}} &= mod(\tilde{\alpha} \cup \tilde{\mathcal{A}}, \mathcal{O}) && \text{(Def. } \mathcal{M}_\alpha^{\mathcal{A}}\text{)} \\
&\subseteq mod(\tilde{\mathcal{M}} \cup \tilde{\alpha} \cup \tilde{\mathcal{A}}, \mathcal{O}) && \text{(Monotonie)} \\
&= mod(\tilde{\mathcal{M}} \cup \tilde{\mathcal{A}}, \mathcal{O}) && (\tilde{\alpha} \subseteq \tilde{\mathcal{M}}) \\
&= mod(\tilde{\mathcal{M}}, \mathcal{O}) && (\tilde{\mathcal{A}} \subseteq \tilde{\mathcal{M}}) \\
&= \mathcal{M}
\end{aligned}
$$

Die Korrektheit des nächsten Korollars wird nicht gezeigt, folgt hier entsprechend zu dem Korollar in [Del+19] aus dem vorherigen Lemma und Definitionen.

Korollar 5.9. Für alle $\mathcal{A}$-Atome $\mathfrak{a}, \mathfrak{b} \in \mathfrak{A}_{\mathcal{A}}(\mathcal{O})$ gilt:

1. $\mathcal{M}_{\mathfrak{a}}^{\mathcal{A}} = \mathcal{M}_\alpha^{\mathcal{A}}$ für jedes Axiom $\alpha \in \mathfrak{a}$.
2. $\mathfrak{a}$ hängt von allen Atomen, die in $\mathcal{M}\mathfrak{a}^{\mathcal{A}} \setminus \mathfrak{a}$ enthalten sind, ab.
3. $\mathfrak{a} \succeq \mathfrak{b}$ genau dann, wenn $\mathcal{M}_{\mathfrak{a}}^{\mathcal{A}} \supseteq \mathcal{M}_{\mathfrak{b}}^{\mathcal{A}}$
4. Für jedes Axiom $\beta \in \mathcal{O} \setminus \mathcal{A}$ gilt, dass $\beta \in \mathfrak{a}$ genau dann, wenn $\mathcal{M}_\beta^{\mathcal{A}} = \mathcal{M}_{\mathfrak{a}}^{\mathcal{A}}$.
5. Für alle Axiome $\alpha, \beta \in \mathcal{O} \setminus \mathcal{A}$ gilt, dass $\mathcal{M}_\alpha^{\mathcal{A}} = \mathcal{M}_\beta^{\mathcal{A}}$ genau dann, wenn es ein Atom gibt, sodass $\{\alpha, \beta\} \subseteq \mathfrak{b}$
6. Es gilt $\mathfrak{a} = \{\alpha | \mathcal{M}_\alpha^{\mathcal{A}} = \mathcal{M}_{\mathfrak{a}}^{\mathcal{A}}\}$

Algorithmus 3 zeigt in Pseudocode auf, wie die erweiterte atomare Dekomposition berechnet werden kann. Dieser ist eine leicht abgeänderte Version des Algorithmus zur Berechnung der atomaren Dekomposition aus [Del+19]. Der Algorithmus initialisiert zunächst eine Menge *KeyAxs* für Axiome, welche jeweils zu einem Atom gehören und eine Menge *TauAtom* für gefundene, offensichtliche Tautologien. Anschließend wird für jedes noch nicht gewählte Axiom das kleinste Modul berechnet, welches dieses und die bereits gewählten Axiome enthält. Danach wird überprüft, ob das entstandene Modul leer ist und es sich bei dem Axiom folglich um eine offensichtliche Tautologie handelt. Ist dies nicht der Fall, wird für jedes bereits gewählte *KeyAxs* überprüft, ob es sich bei dem Modul des neuen Axioms um dasselbe Modul handelt und somit beide Axiome zu einem Atom gehören. Ist das neue Axiom in keinem der Module enthalten, dann wird es selbst ein Bestandteil von *KeyAxs*. Nachdem dies für alle nicht gewählten Axiome durchgeführt wurde, wird die Abhängigkeitsrelation berechnet. Dafür wird für jedes Paar (α,β) von *KeyAxs* betrachtet, ob α im Modul von β enthalten ist. In diesem Fall ist α von β abhängig. Abschließend wird die so erhaltene eAD zurückgegeben. Die Korrektheit der *KeyAxs* folgt aus Korollar 5.9.3. Die Bestimmung der Abhängigkeiten ist auf Grund von Lemma 5.8 und Korollar 5.9.4 korrekt. Die Laufzeit dieses Algorithmus unterscheidet sich nicht von der in [Del+19] beschriebenen Laufzeit für die atomare Dekomposition.

5.5.1 Extraktion mit erweiterter atomarer Dekomposition

Wie bereits angedeutet, lässt sich die erweiterte atomare Dekomposition für einen verbesserten Extraktionsprozess verwenden. Dieser ist in Algorithmus 4 dargestellt. Den Nutzenden wird dabei jeweils die erweiterte atomare Dekomposition der Ontologie in

Input : Eine Ontologie O, eine Funktion x-Modul(*,*) zur Berechnung von Modulen bezüglich einer Signatur, A (Menge der bereits gewählten Axiome)
Output : Erweiterte atomare Dekomposition bezüglich A
$KeyAxs \leftarrow \emptyset$
$TauAtom \leftarrow \emptyset$
for ***each*** $\alpha \in \mathcal{O} \setminus A$ **do**
  $Module(\alpha) \leftarrow x\text{-}mod(\widetilde{\alpha} \cup \widetilde{A}, \mathcal{O})$
  if $Module(\alpha) = \emptyset$ **then**
    $TauAtom \leftarrow TauAtom \cup \{\alpha\}$
  else
    $new \leftarrow true$
    for ***each*** $\beta \in KeyAxs$ **do**
      if $Module(\alpha) = Module(\beta)$ **then**
        $Atom(\beta) \leftarrow Atom(\beta) \cup \{\alpha\}$
        $new \leftarrow false$
        $KeyAxs \leftarrow KeyAxs \cup \{\alpha\}$
$dep \leftarrow \emptyset$
for ***each*** $\alpha \in KeyAxs$ **do**
  for ***each*** $\beta \in KeyAxs$ **do**
    if $\beta \in Module(\alpha)$ **then**
      $dep \leftarrow dep \cup \{(Atom(\beta), Atom(\alpha))\}$
$Atom \leftarrow \{Atom(\alpha) | \alpha \in KeyAxs\}$
$GenMods \leftarrow \{Module(\alpha) | \alpha \in KeyAxs\}$
return [*GenMods,(Atoms,dep),KeyAxs*]

Algorithmus 3 : Extraktion eines Moduls basierend auf atomarer Dekomposition

Abhängigkeit der bereits gewählten Axiome aus M als Graph angezeigt. In dieser können Nutzende ein Atom wählen, welches in die Menge der gewählten Axiome aufgenommen wird. Nach der Aufnahme neuer Axiome in M muss die erweiterte atomare Dekomposition erneut berechnet werden.

5.5.2 Fallbesipiel

An einem Beispiel soll nun gezeigt werden, wie die Wahl eines Moduls mithilfe der erweiterten atomaren Dekomposition durchgeführt werden kann. Dazu wird die Koala 2 genannte Ontologie aus [Del+19] verwendet. Diese ist eine verkleinerte Version der bekannten Koala Ontologie.
Nehmen wir als Anwendugsfall an, dass ein/e Entwickler*in eine Ontologie über heimische Tiere in den Wäldern der Alpen erstellen möchte. Dabei ist er/sie auf die Koala 2 Ontologie gestoßen, von welcher er/sie vermutet, dass diese für ihn/sie relevantes Wissen enthalten könnte, welches wiederverwendbar sein könnte.
Daher versucht er/sie dieses Wissen mithilfe der erweiterten atomaren Dekomposition zu finden und zu extrahieren. Dafür visualisiert ihm/ihr das Tool zunächst die erwei-

Input : Eine Ontologie O, eine Funktion x-mod(*,*) zur Berechnung von Modulen bezüglich einer Signatur
Output : x-Modul M aus O
$M = \emptyset$
repeat
  Graph G = berechne erweiterte atomare Dekomposition basierend auf Modulart x bezüglich M.
  Visualisiere G
  Nutzer wählt Atom a in G
  M' =x-Modul($\tilde{a}, \mathcal{O}$))
  $M = M'$
until *Nutzer beendet Wahl von Axiomen*
return M

Algorithmus 4 : Extraktion eines Moduls basierend auf erweiterter atomarer Dekomposition

terte atomare Dekomposition ohne bereits gewählte Axiome. Diese entspricht der nicht erweiterten atomaren Dekomposition von Koala 2 und ist in Abbildung 5.1 dargestellt. Zur Beschriftung der Atome wird in den folgenden Abbildungen jeweils die eingeführte Singnaturerweiterung genutzt.

In dieser findet der/die Entwickler*in das Konzept Animal. Diese weckt sein/ihr Interesse, da er/sie in seiner/ihrer Ontologie Aussagen über Tiere treffen möchte. Anhand der Abhängigkeitsbeziehungen kann überprüft werden, welche Konzepte und Rollen in einem Modul, welches Animal enthält, noch enthalten wären. Daraus kann er/sie schließen, welche Wissensgebiete über Tiere in der Ontologie enthalten sind. In diesem Fall handelt es sich um Lebensräume und Geschlechter von Tieren. Angenommen der/die Entwickler*in ist mit diesen Informationen zufrieden, wählt er/sie das Konzept Animal. Das Tool speichert alle durch diese Wahl im Modul enthaltenen Axiome als gewählte Axiome. Anschließend wird die erweiterte atomare Dekomposition unter Berücksichtigung der bereits gewählten Axiome berechnet und angezeigt (dargestellt in Abbildung 5.2). In dieser findet der/die Entwickler*in anschließend das Konzept Forest, welches er wiederum wählt. Die Menge der gewählten Axiome wird erweitert und wiederum die erweiterte atomare Dekomposition berechnet (dargestellt in Abbildung 5.3). In dieser findet der/die Entwickler*in keine weiteren Konzepte oder Rollen mehr, welche er/sie für seine/ihre Ontologie übernehmen möchte. Damit ist der Extrakionsprozess erfolgreich abgeschlossen und die Menge der gewählten Axiome kann als Modul zurückgegeben werden.

Dieses Beispiel ist nicht repräsentativ für den Prozess der Modulextraktion. Die Koala 2 Ontologie ist eine extrem kleine Ontologie und der Graph der erweiterten atomaren Dekomposition ist daher zu jedem Zeitpunkt sehr übersichtlich. Es lässt sich aus diesem Beispiel noch nicht schließen, wie sich die erweiterte atomare Dekomposition auf größere

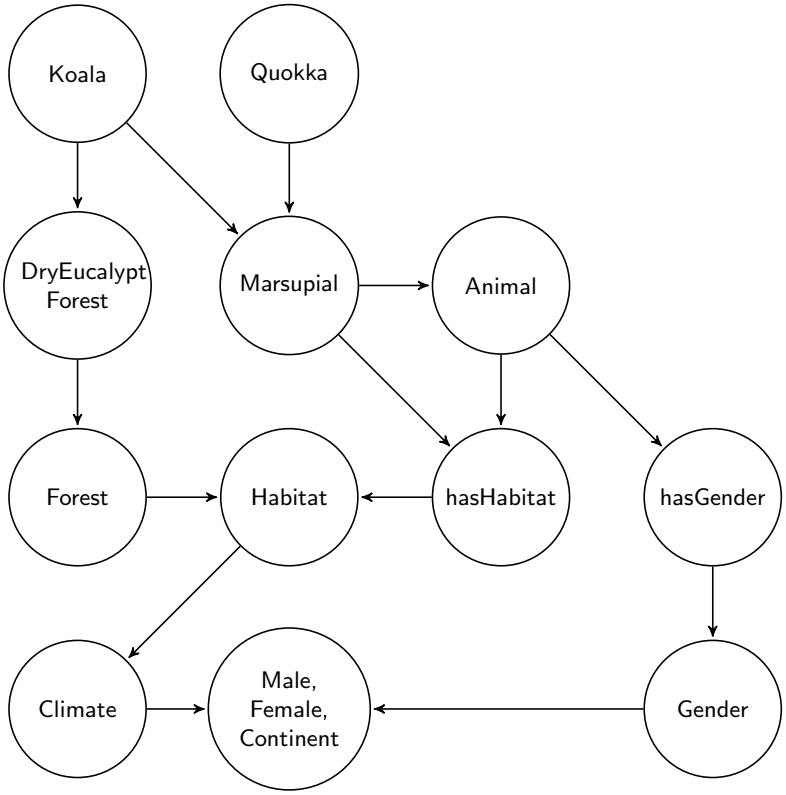

Abbildung 5.1 eAD von Koala 2 ohne bereits gewählte Atome

Ontologien anwenden lässt.

5.5.3 Implementation

Um die Strukturen der Graphen, welche bei der erweiterten atomaren Dekomposition entstehen, bewerten zu können, wurde für diese Arbeit eine Beispielimplementation angefertigt. Diese basiert auf der Implementation der atomaren Dekomposition in den OWL-API Tools[1]. Diese musste, wie es schon der Pseudocode vermuten lässt, um die Menge bereits gewählter Axiome erweitert werden. Zusätzlich wurde eine Visualisierung für diese Implementation erstellt, mit der sich Atome in der erweiterten atomaren Dekomposition wählen lassen. Die Atome werden in der Visualisierung mit den enthaltenden Axiomen beschriftet. Als Sprache wurde Java verwendet, da die OWL-API Tools in Java verfasst sind.
Für die Visualisierung des Graphen wurde ein passendes Framework gesucht. Anforderung an dieses war vor allem, eine beliebige Graphstruktur automatisch visualisieren zu können, ohne aufwendig die Position jedes Knotens bestimmen zu müssen. Eingesetzt wurde GraphStream, welches zunächst sehr hilfreich war, da es mit minimalem Aufwand eine akzeptabele Visualisierung ermöglichte. Während der weiteren Implementation stell-

[1] *https://github.com/owlcs/owlapi/tree/version5/tools*

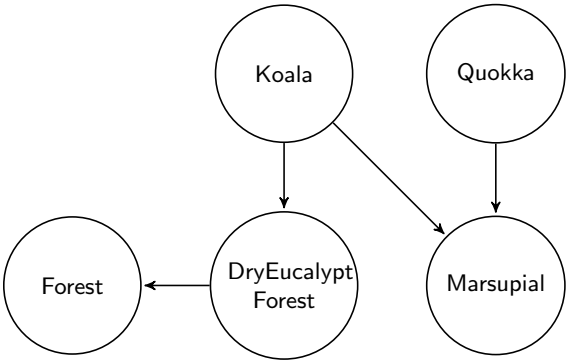

Abbildung 5.2 eAD von Koala 2 nach der Wahl von Animal

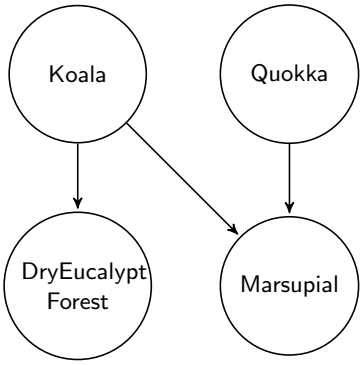

Abbildung 5.3 eAD von Koala 2 nach der Wahl von Animal und Forest

te sich jedoch heraus, dass einige Einschränkungen bestehen. So ist der Einfluss auf das Layout des Graphen nicht sonderlich groß, was in einigen ungewünschten Phänomenen resultiert. Dazu gehören eine schwer kontrollierbare Kantenlänge sowie überlappende Knoten und eine schlechte Beschriftung der Knoten. Um das Fallbeispiel in Kapitel 5.5.2 zu ermöglichen und die theoretischen Nach- und Vorteile der Extraktion mit erweiterter atomarer Dekomposition in der Realität zu beobachten, genügt die Implementation jedoch.

5.5.4 Bewertung der Modulextraktion mit erweiterter atomarer Dekomposition

Nachvollziehbarkeit Die Nachvollziehbarkeit, welche Atome durch die Wahl eines Atoms noch in das Modul gelangen, ist bei dem beschriebenen Verfahren gut. Es kann den Abhängigkeiten des gewählten Atoms gefolgt werden. Dies gelingt, im Vergleich zur atomaren Dekomposition, auch wenn bereits mehrere Atome gewählt wurden.

Granularität Die Granularität hängt, wie bei der atomaren Dekomposition, von der gewählten Modulart ab.

Verständlichkeit der Teile Bei der Verständlichkeit der Teile gibt es keinen Unterschied zu der Verständlichkeit der Teile der atomaren Dekomposition.

Skalierbarkeit der Visualisierung Bei steigender Größe der Ontologie steigt, wie bei der atomaren Dekomposition, die Komplexität des Graphen. Vorteilhaft ist jedoch, dass der Graph mit jedem gewählten Atom schrumpft.

Skalierbarkeit der Berechnung Initial und nach jeder Wahl eines Atoms muss die erweiterte atomare Dekomposition berechnet werden. Dies benötigt, wie schon die atomare Dekomposition, in Abhängigkeit von der Größe der Ontologie linear viele Modulextraktionen. Die Verzögerung für das initiale Laden und das Aktualisieren nach der Wahl eines Atoms sind somit gleich aufwendig. Damit ist die Aktualisierung aufwändiger als für die atomare Dekomposition.

Fazit Insgesamt lässt sich zusammenfassen, dass die erweiterte Dekomposition das Problem der Wahl mehrere Atome in ein Modul löst, jedoch die anderen Nachteile der atomaren Dekomposition übernimmt. Außerdem ist die Berechnung schlechter skalierbar.

5.5.5 Alternative Ansichten auf die atomare Dekomposition

Da, wie zuvor beschrieben, die Komplexität des Graphen der erweiterten atomaren Dekomposition eines der Hauptprobleme ist, sollen in diesem Abschnitt zwei Ansichten auf den Graphen erarbeitet werden, welche es Nutzenden ermöglicht, gezielt nach bestimmten Erweiterungen für ein Modul zu suchen oder alternativ minimale Erweiterungen für das Modul zu betrachten.

Iterative Wahl Damit ein Modul möglichst iterativ gewählt werden kann, könnte es sinnvoll sein, dieses immer nur minimal zu erweitern. Um dies umsetzen zu können, muss zunächst erkannt werden, dass es Atome gibt,welche das Modul kleinstmöglich erweitern.

Definition 5.10. Ein Atom, welches von keinen anderen Atomen abhängig ist, ist ein minimales Atom.

Es ist leicht zu erkennen, dass es immer solche minimalen Atome geben muss. Außerdem muss jedes beliebige Atom, welches kein minimales Atom ist, von mindestens einem minimalen Atom abhängig sein. Somit wird, unabhängig davon, welches Atom gewählt wird, immer mindestens ein minimales Atom mitgewählt. Werden nur die minimalen Atome zur Wahl gestellt (z.B. als Liste, da diese keine Abhängigkeiten untereinander haben), wird nicht beschränkt, welche Module mit der erweiterten atomaren Dekomposition gewählt werden können. Die Wahl eines Atoms mit Abhängigkeiten kann simuliert werden, indem das Hauptideal des Atoms nach und nach gewählt wird. Unpraktisch ist diese Ansicht, wenn nach einem bestimmten Atom bzw. Element in dessen Beschriftung

gesucht wird. Taucht dieses nicht in den minimalen Atomen auf, gibt es keine Möglichkeit, zu erkennen, welche minimalen Atome gewählt werden müssen, damit sich das gesuchte Atom wählen lässt.

Suchansicht Sind Nutzende an bestimmtem Vokabular der Ontologie interessiert, bietet sich eine Suchfunktion an. Diese kann auf Grundlage der in dieser Arbeit eingeführten Signaturerweiterung umgesetzt werden. Ein Element der Signatur der Ontologie kommt in mindestens einer Signaturerweiterung eines Atoms der erweiterten atomaren Dekomposition vor, sofern das Element nicht bereits in der Signatur des aktuell gewählten Moduls ist. Für die Suche nach einem Element sollten somit alle Atome angezeigt werden, welche das Element in ihrer Signaturerweiterung enthalten. Die Suche kann dabei mit einer Suchleiste umgesetzt werden und die Ergebnisse können wieder als Liste präsentiert werden. Um eine Erweiterung für ein Modul zu finden, ohne spezielles Vokabular zu suchen, eignet sich diese Ansicht nicht, hierfür wurde jedoch bereits die inkrementelle Ansicht eingeführt.

Die beiden Ansichten sind weniger komplex darzustellen als der Graph, da die Atome jeweils in einer Liste untereinander dargestellt werden können. Außerdem ist davon auszugehen, dass die Menge aller minimalen Atome bzw. die Menge aller Atome zu einem Suchbergriff für gewöhnlich deutlich kleiner sind als die Gesamtzahl der Atome.

5.6 Fazit

Als vielversprechender Ansatz zur Wahl eines Moduls hat sich in diesem Kapitel die erweiterte atomare Dekomposition herausgestellt. Diese bietet Nutzenden die Möglichkeit, ein Modul anhand des Inhaltes zu wählen. Das Problem der Beschriftung von Atomen wurde für diesen Anwendungsfall gelöst und außerdem wurden drei Visualisierungen der erweiterten atomaren Dekomposition erarbeitet. Diese beinhalten die offensichtliche Darstellung als Graph, eine Liste zur inkrementellen Wahl von minimalen Atomen und eine Suchansicht, welche passende Atome zu einem Suchbegriff liefert. Interessant macht den Ansatz des Weiteren, dass dieser bis jetzt in keiner Anwendung umgesetzt wurde.

Kapitel 6

Umsetzung eines Ansatzes

In diesem Kapitel soll die Umsetzung des Ansatzes behandelt werden. Zunächst wird dazu ein Ansatz ausgewählt. Danach werden die Anforderung an die Anwendung erarbeitet, Software ausgewählt und anschließend die Implementation erläutert. Mit einer aus mehreren Ontologien bestehenden Fallstudie wird diese anschließend untersucht. Anhand dieser Erfahrungen wird die Anwendung untersucht, Probleme erörtert und Erweiterungsmöglichkeiten diskutiert. Zuletzt werden die Haupterkenntnisse aus diesem Kapitel im Fazit zusammengefasst.

6.1 Auswahl eines Ansatzes

Im Rahmen dieses Abschnitts soll ein Ansatz zur Extraktion eines Moduls aus einer Ontologie als Tool beispielhaft umgesetzt werden. Nachdem sowohl für die signaturbasierten Verfahren als auch für die dekompositionsbasierten Verfahren ein bester Ansatz bestimmt worden ist, muss final entschieden werden, welcher dieser beiden Ansätze umgesetzt wird. Um diesen Vergleich durchführen zu können, folgt zunächst eine kurze Zusammenfassung der jeweiligen Ansätze:

Hybride aus eingerückter Liste und Graphen

- Parallele Darstellungen der Elemente der Signatur durch eingerückte Listen und Graphen
- Darstellungen von Konzept und Rollenhierarchie in eigenen Ansichten
- Anreicherungen des Graphen der Konzepthierarchie durch Verwendung von Rollenbeziehungen
- Bei iterative Wahl aktuell im Modul befindliche Elemente markieren

Iterative Wahl in der erweiterten atomaren Dekomposition

- Beschriftung der Atome durch Erweiterung der Signatur
- Unterschiedliche Sichtweisen auf erweiterte atomare Dekomposition möglich
- Ansicht als Graph für Gesamtübersicht

- Auswahlmöglichkeiten, um ein Element in die Signatur aufzunehmen
- Inkrementelle Wahl, um mögliche Erweiterungen für ein Modul zu finden

Zunächst lässt sich argumentieren, dass beide Ansätze mit Graphen arbeiten und diese in beiden Fällen nur schwer skalierbar sind. Daher bringen beide Ansätze alternative Ansichten mit sich, welche verwendet werden können, wenn die Graphen zu unübersichtlich werden. Dies wäre für den signaturbasierten Ansatz die Ansicht als eingerückte Liste und für die erweiterte atomare Dekomposition die Kombination aus Suchansicht und inkrementeller Ansicht.

Im Allgemeinen besteht ein Hauptunterschied in den visualisierten Informationen. Während Hybride aus eingerückter Liste und Graphen versuchen, die inhaltlichen Zusammenhänge zwischen den Elementen bestmöglich darzustellen, verfolgt die iterative Wahl in der erweiterten atomaren Dekomposition vor allem das Ziel, Abhängigkeiten zwischen Teilen der Ontologie im Bezug auf Module darzustellen.

Ob es für Anwendende im Prozess der Modulextraktion sinnvoller ist, mit Informationen über das Vokabular der Ontologie zu arbeiten und eine Schnittstelle zum Modul zu definieren, wie es bei signaturbasierten Verfahren der Fall ist, oder, ob es sinnvoller ist, mit Informationen über Zusammenhänge von Vokabular im Bezug auf mögliche Module zu arbeiten, wie es bei dem vorgestellten dekompositionsbasierten Ansatz der Fall ist, kann an dieser Stelle nicht abschließend bewertet werden. Es lässt sich jedoch argumentieren, dass sich Anwendende, die sich mit der Ontologie auskennen bzw. die Bedeutung und Zusammenhänge des Vokabulars kennen, an Informationen über mögliche Module interessiert sein werden. Somit ließe sich die Spezifikation eines Moduls in zwei Phasen aufteilen. In der ersten setzen sich die Nutzenden mit der Ontologie und ihrem Vokabular auseinander. Dazu können singaturbasierte Visualisierung aus in dieser Arbeit vorgestellten Anwendungen zur Hilfe genommen werden. In der zweiten Phase wird mithilfe einer dekompositonsbasierten Visualisierung die Spezifikation durchgeführt. Hierfür gibt es jedoch aktuell keine Anwendung. Daher wird diese im Folgenden entwickelt. Eine Anwendung könnte auch beide Phasen unterstützen, dies ist jedoch für den Rahmen dieser Arbeit zu umfangreich.

6.2 Anforderung an das Tool

Begonnen wird in diesem Abschnitt mit der Formulierung der Anforderungen an das Tool. Zunächst werden Anforderungen genannt, welche notwendig sind, damit der Prozess der Modulextraktion durchführbar ist. Anschließend folgen Anforderungen, welche den Prozess für die Nutzenden vereinfachen. Für einzelne Ansichten folgen abschließend noch einige spezifische Anforderungen und es wird erläutert, welche Performance dabei benötigt wird.

Um mit dem Prozess der Modulextraktion beginnen zu können, muss die Anwendung das Laden von Ontologien, welche in OWL verfasst wurden, ermöglichen. Für eine geladene Ontologie wird von der Anwendung erwartet, die erweiterte atomare Dekomposition berechnen zu können. Damit ein Modul spezifiziert werden kann, muss es den Nutzenden möglich sein, Atome aus der Dekomposition zu wählen, welche dann in das Modul aufgenommen werden. Nach der Wahl eines Atoms sollte die erweiterte atomare Dekomposition in Abhängigkeit der gewählten Atome erneut berechnet werden. Um ein auf diese Weise spezifiziertes Modul nutzbar zu machen, sollte die Anwendung in der Lage sein, nach der Wahl einer Menge von Atomen das zugehörige Modul zu bestimmen. Für die Weiterverwendung des Moduls ist von Bedeutung, dass dieses gespeichert werden kann.

Um die Nutzenden der Anwendung bei der Wahl eines Moduls zu unterstützen, sollten die drei Sichten auf die erweiterte atomare Dekomposition umgesetzt werden. Außerdem ist der Wechsel zwischen den Ansichten während der Spezifikation des Moduls sinnvoll. Des Weiteren kann das aktuell gewählte Modul dargestellt werden, damit den Nutzenden zu jedem Zeitpunkt erkenntlich ist, was im aktuellen Modul enthalten wäre. Wenn sich die Wahl eines Atoms nach dessen Wahl als fehlerhaft erweist, sollte es möglich sein, diese Wahl rückgängig zu machen, ohne den gesamten Prozess von vorne beginnen zu müssen. Da in allen Ansichten Atome angezeigt werden, ist es nötig, dass diese für die Nutzenden leicht verständlich beschriftet sind.

Neben diesen Gesamtanforderungen sollten auch Anforderungen an die einzelnen Ansichten gestellt werden. Von der Graphansicht wird zunächst gefordert, den gesamten Graph anzuzeigen. Außerdem sollten sich Knoten/Atome nicht überlappen und es sollte möglich sein, einzelnen Kanten im Graphen zu folgen. Wenn ein Atom im Graphen gewählt wird, dann ist es sinnvoll, zu visualisieren, welche Atome durch dessen Wahl mit im Modul aufgenommen werden.

In der Suchansicht muss es vor allem möglich sein, ein Element zu suchen und für ein gesuchtes Element jede minimale Möglichkeit, dieses in das Modul aufzunehmen, angezeigt zu bekommen. Die inkrementelle Ansicht hingegen muss den Nutzenden zu jedem Zeitpunkt alle Atome zur Wahl stellen, welche, ohne andere Atome hinzuzunehmen, in das Modul aufgenommen werden können.

Bei den Anforderungen an die Performance sind vor allem die Momente zu betrachten, in welchen die erweiterte atomare Dekomposition berechnet werden muss. Gut wäre, wenn die resultierende Verzögerung weniger als 10 Sekunden betragen würde. Als akzeptabel kann eine Verzögerung von bis zu einer Minute betrachtet werden, da die Berechnung jeweils nach jeder Erweiterung durchgeführt wird und längere Wartezeiten diesen iterativen Prozess stark behindern würden.

6.3 Software von Auswahl

Um ein Tool umzusetzen, welches die Wahl eines Moduls über die iterative Wahl in der erweiterten atomaren Dekomposition umsetzt, muss zunächst eine Entscheidung über die verwendete Programmiersprache und die verwendeten Frameworks getroffen werden. Zunächst stand fest, dass die OWL API verwendet werden sollte, um mit OWL-Ontologien zu arbeiten. Außerdem sollte die in Kapitel 5.5.3 vorgestellte Implementation der erweiterten atomaren Dekomposition auf Grundlage der OWL API Tools weiterverwendet werden. Damit wäre es am einfachsten, auch im Rest der Implementation auf Java zu setzen. Wie in Kapitel 5.5.3 ausgeführt, ist das dort verwendete Framework für die Visualisierung von Graphen jedoch nicht zufriedenzustellend. Daher wurde für die Entwickelung des Tools auf ein anderes Framework gesetzt. Dieses heißt JGraphX und ist für die Integration in Oberflächen, welche in Swing geschrieben wurden, vorgesehen.

6.4 Umsetzung

Im folgenden Abschnitt werden einige Einblicke in die Umsetzung gewährt. Dabei wird zunächst beschrieben, wie die erweiterte atomare Dekomposition verwaltet wird, sodass mehrere Ansichten abwechselnd von dieser Gebrauch machen können. Anschließend wird beschrieben, wie die einzelnen Ansichten umgesetzt wurden. Ein Klassendiagramm der Implementation zusammen mit Kurzbeschreibungen zu den einzelnen Klassen findet sich im Anhang.

Synchronisierung der Ansichten Um alle Ansichten zu synchronisieren, damit während der Wahl eines Moduls zwischen diesen gewechselt werden kann, implementieren alle Ansichten ein Interface mit dem Namen „eadView “. Dieses stellt die Methode updateUI() bereit, welche dafür vorgesehen ist, die Ansicht an die Änderungen der erweiterten atomaren Dekomposition anzupassen. Dabei wird die erweiterte atomare Dekomposition basierend auf $\bot$-Modulen berechnet. Eine solche Änderung kann geschehen, wenn ein Atom gewählt wird oder eine Wahl rückgängig gemacht wird. Bei der Initialisierung wird ein Objekt der Klasse CurrentState erstellt, welches dazu dient, den aktuellen Stand der Extraktion samt gewählter Axiome und aktueller atomarer Dekomposition zu verwalten. Anschließend werden mit diesem Objekt die einzelnen Ansichten erstellt. Diese registrieren sich beim CurrentState mit der addView()-Methode. Werden Änderungen am CurrentState vorgenommen, können mit dem Aufruf der updateViews()-Methode alle Ansichten angepasst werden.

Hauptmenü Das Hauptmenü besteht aus zwei Bestandteilen. Zum einen befindet sich links ein Reiter, mit dem sich zwischen den Ansichten umschalten lässt und darunter die aktuelle Ansicht. Zum anderen wird rechts das aktuelle Modul dargestellt. Dies erfolgt

über die Signatur des Moduls. Diese wird unterteilt in Konzepte, Rollen und Individuen angezeigt. Unter der Signaturanzeige befinden sich noch zwei Knöpfe, welche aktiviert werden, sobald ein Modul gewählt wurde. Einer macht die letzte Änderung rückgängig und der andere ermöglicht das Öffnen des Dialogs zum Speichern des Moduls. Zum Start der Applikation befindet man sich im Hauptmenü und der Ansichtsreiter ist in der Ontologieansicht. Alle anderen Ansichten sind zunächst gesperrt und werden erst nach dem Laden einer Ontologie freigeschaltet. Abbildung B.1 zeigt das Hauptmenü.

Graphansicht In der Graphansicht wird die aktuelle erweiterte atomare Dekomposition als Graph angezeigt. Da die erweiterte atomare Dekomposition ein kreisfreier Graph ist, kann für dessen Layout ein Algorithmus zum Anordnen von hierarchischen Graphen verwendet werden. Einen solchen stellt das Framework JGraphX zur Verfügung. Der angezeigte Graph enthält für jedes Atom einen Knoten, welcher mit der Signaturerweiterung beschriftet ist und Kanten für die Abhängigkeitsrelation zwischen den Atomen enthält. Den Nutzenden ist es nicht möglich, diesen Graphen zu verändern. Ein Beispiel für die Graphansicht ist in Abbildung B.2 dargestellt.

Suchansicht In der Suchansicht werden die Nutzenden aufgefordert, in einer Eingabeleiste das Element einzugeben, welches sie der Modulsignatur hinzufügen möchten. Dabei ist nur die Eingabe gültiger Namen möglich und die Liste aller Elemente kann betrachtet werden. Für ein aktuell ausgewähltes Element gibt es zwei Optionen. Entweder es gibt nur ein einziges Atom, welches das Element der Signatur hinzufügt oder es gibt mehrere. Im ersten Fall wird angezeigt, welche zusätzlichen Elemente neben dem gesuchten Element zur Signatur hinzugefügt werden müssen, damit das Element der Signatur hinzugefügt werden kann. Einen solchen Fall zeigt Abbildung B.4. Gibt es mehrere Möglichkeiten, werden links verschiedene Möglichkeiten angezeigt, die Signatur zu erweitern und rechts Elemente, welche in allen diesen Möglichkeiten enthalten sind. Dies ist in Abbildung B.3 zu sehen.

Inkrementelle Ansicht Die inkrementelle Ansicht ist die am einfachsten aufgebaute Ansicht. Sie besteht aus einer Liste von Atomen, welche von keinen anderen Atomen abhängig sind. Beschriftet sind die Listeneinträge, wie auch im Graphen, mit der Singnaturerweiterung. Den Nutzenden ist es möglich, ein beliebiges Atom aus dieser Liste zu wählen und dem Modul hinzuzufügen. Eine solche Liste zeigt Abbildung B.5.

6.5 Fallstudien mit Beispielontologien

In diesem Abschnitt wird das Tool beispielhaft auf vier Ontologien angewendet. Diese unterscheiden sich unter anderem in ihrem Umfang. Eine Übersicht bietet Tabelle 6.1. Zunächst werden die beiden in dieser Arbeit verwendeten Beispielontologien Koala

und Pizza betrachtet. Als erste Ontologie mit relevantem Inhalt folgt die Biomedical Resource Ontology(BRO). Diese hat deutlich mehr Konzepte als Pizza, die Anzahl an logischen Axiomen ist jedoch geringer als in Pizza. Zuletzt folgt die Radiology Lexicon(RadLex) Ontologie. Die Software Ontology(SWO) beschäftigt sich mit Software, welche im Kontext der Bioinformatik eine Rolle spielt. Sie hat mit 2124 logischen Axiomen beinahe vierfach so viele logische Axiome wie Pizza. Mit 7990 logischen Axiomen wiederum deutlich größer ist die EDAM Ontolige, welche übliche Konzepte aus der Bioinformatik beschreibt. Die Ontologie über Radiologie ist um ein vielfaches umfangreicher als die anderen. Jede dieser Ontologien wird im Folgenden mit dem Tool geladen und in allen drei Ansichten betrachtet. Dabei wird untersucht, wie schnell die Ontologien geladen werden und wie sich die Ansichten verhalten.

Ontologie	Anzahl Konzepte	Anzahl logische Axiome	Zeit zur Berechnung der EAD in Sekunden
Koala	21	41	0,065
Pizza	100	712	4,883
BRO	488	560	1,170
SWO	728	2124	4,609
EDAM	3280	7990	43,412
RadLex	46636	128731	Timeout(5min)

Tabelle 6.1 Gegenüberstellung verwendeter Ontologien

Koala Das Laden von Koala und das Berechnen der initialen erweiterten atomaren Dekomposition geschieht nahezu verzögerungsfrei. Die Graphansicht von Koala wurde bereits in Abbildung B.2 dargestellt. Auf einem durchschnittlich großen Laptopbildschirm lässt sich der gesamte Graph betrachten. Dabei ist es möglich, jede Kante zu verfolgen. In der Suchansicht wird für gesuchte Elemente eine übersichtliche Erweiterung der Modulsignatur angezeigt. Für die meisten Elemente gibt es keine Auswahl an unterschiedlichen Erweiterungen. Für das Konzept Habitat gibt es jedoch zwei unterschiedliche Erweiterungen. Die inkrementelle Anzeige bietet, sofern noch kein Atom gewählt wurde, zunächst nur ein Atom zur Wahl. (Dieses ist somit in allen Modulen enthalten.) Nach der Wahl dieses Atoms kann zwischen mehreren Atomen gewählt werden. Die Anzahl der wählbaren Atome bleibt hingegen übersichtlich.

Pizza Die Pizza-Ontologie kann mit dem Tool geladen werden, das Berechnen der erweiterten atomaren Dekomposition dauert jedoch wenige Sekunden. Die Graphansicht ist hingegen kaum noch nutzbar. Während die Tiefe des Graphen nicht viel größer ist als die Tiefe des Graphen von Koala, erstreckt sich die Breite über viele Bildschirme. Außerdem

enthält der Graph so viele Kanten, dass sich eine einzelne Kante kaum verfolgen lässt. Auffällig ist auch, dass viele der Atome als Beschriftung die leere Menge haben. Diese repräsentieren somit Module, deren Signatur eine Vereinigung mehrerer anderer Module sind. In der Suchansicht gibt es für viele der Elemente genau eine Signaturereiterung, um dieses in die Signatur des Moduls aufzunehmen. Die Menge der Elemente, welche durch die Wahl eines Elements automatisch mitgewählt werden, ist hier jedoch bereits deutlich größer. Die inkrementelle Ansicht beginnt wieder mit einem Atom, welches in allen nichtleeren Modulen enthalten ist. Anschließend ist meist die Wahl verschiedener Atome möglich. Dabei ist die Anzahl der gleichzeitig angezeigten Atome zwar mitunter schon größer, kann jedoch überblickt werden.

BRO Die Biomedical Resource Ontology verhält sich im Bezug zum Laden und Berechnen der erweiterten atomaren Dekomposition ähnlich wie Pizza. Des Weiteren wird auch hier der Graph sehr breit und nicht besonders tief. Allerdings fällt hier zusätzlich auf, dass es ein überdurchschnittlich großes Atom gibt, von dem alle anderen Atome abhängen. Spannend ist auch, dass nach der ersten Wahl, welche zwangsläufig das große Atom enthält, der Graph in mehrere unzusammenhängende Teilgraphen zerfällt. Einige dieser sind klein und leicht zu überblicken. Die für Pizza beschriebenen Atome mit leerer Beschriftung gibt es hier nicht. In der Suchansicht zeigt sich, neben vielen Elementen, die erneut nur eine Wahlmöglichkeit haben, mit dem Konzept Activity ein Element, für welches es 14 unterschiedliche Wahlmöglichkeiten gibt. In der inkrementellen Ansicht muss logischerweise zunächst das beschriebene große Atom ausgewählt werden. Anschließend steht eine Menge an Atomen zur Auswahl, welche nicht auf einem Bildschirm dargestellt werden kann.

SWO und EDAM Während sich die Software Ontology in 4,6 Sekunden laden lässt, benötigt die EDAM Ontologie 43 Sekunden. Für beide Ontologien ist die Anwendung jedoch nicht nützlich. Dies liegt daran, dass die Konzeptnamen in diesen Ontologien (ohne genauere Beschreibung) nicht aussagekräftig sind.

RadLex Das Laden der RadLex Ontologie ist möglich, aber die erweiterte atomare Dekomposition kann nicht in absehbarer Zeit berechnet werden.

6.6 Evaluation

Welche Probleme das beschriebene Tool aufzeigt, welche Verbesserungsmöglichkeiten es noch gäbe und wie einsatzfähig dieses Tool in der Praxis wäre, soll anhand von Beobachtungen aus dem vorherigen Abschnitt betrachtet werden. Dabei werden zunächst Probleme, welche durch die Fallstudien erkannt wurden, erörtert. Anschließend wird auf

Möglichkeiten der Verbesserung eingegangen. Abschließend wird untersucht, wie einsatzfähig das Tool sein könnte.

Erreichte Anforderungen

Die Fallstudien zeigen, dass es mit dem Tool für Ontologien begrenzter Größe möglich ist, diese zu laden, Module zu definieren und diese zu extrahieren. Außerdem können Module anschließend gespeichert werden. Dabei kann die Ontologie in den drei für die erweiterte atomare Dekomposiion vorgesehenen Ansichten betrachtet werden. Zwischen den Ansichten kann dabei gewechselt werden. Des Weiteren ist durch die Darstellung der Signatur das aktuell gewählte Modul zu jedem Zeitpunkt repräsentiert. Somit sind die in Abschnitt 6.2 aufgeführten Anforderungen an die zu entwickelnde Anwendung umgesetzt worden.

Probleme

In diesem Abschnitt werden einige Probleme dargestellt, welche der aktuelle Stand des Tools aufzeigt.

Zunächst ist das mit der Ontologie RadLex beobachtete Verhalten kritisch. Ontologien ab einer gewissen Größe können nicht in absehbarer Zeit geladen werden und die Anwendung ist somit für diese Ontologien nicht nutzbar. Außerdem gilt für Ontologien, welche mit Verzögerungen angezeigt werden, dass es diese Verzögerung nicht nur initial, sondern auch nach jeder Änderung des Moduls gibt, da dann eine neue erweiterte atomare Dekomposition berechnet werden muss.

Ein weiteres Problem ist die schnell steigende Komplexität des Graphen, sodass dieser nicht mehr übersichtlich angezeigt werden kann. In den Fallstudien konnte dazu beobachtet werden, dass die Höhe des Graphen in dem verwendeten Layout-Algorithmus unproblematisch war, die Breite jedoch schnell wuchs. Zusätzlich stellte die Menge der Kanten in der verwendeten Visualisierung ein Problem dar, da diese schnell durch ihre Menge nicht mehr einzeln verfolgbar waren. Das Problem des Graphen ist jedoch von Beginn an eingeplant gewesen und die inkrementelle Sicht und Suchansicht als Alternative vorgesehen gewesen. Trotzdem wäre es wünschenswert, wenn die Graphansicht so lange wie möglich übersichtlich bleibt.

Als problematisch konnte auch identifiziert werden, dass es nicht möglich ist, inhaltliche Zusammenhänge der Elemente zu erfassen. Dabei kann die Konzepthierarchie als Beispiel verwendet werden. Zwar können der erweiterten atomaren Dekomposition Informationen über die Abhängigkeiten zwischen Elementen im Bezug auf ihr Vorkommen in Modulen entnommen werden, es kann jedoch nicht erkannt werden, ob ein Konzept ein Unterkonzept eines anderen ist. Somit setzt diese Repräsentation voraus, dass die Nutzenden bereits im Voraus Wissen über den Inhalt der Ontologie haben oder dieses parallel über eine andere Visualisierung erhalten.

Erweiterungsmöglichkeiten

In diesem Abschnitt werden einige mögliche Erweiterungen für die Anwendung begutachtet. Dies geschieht mit Rücksicht auf die zuvor aufgezeigten Probleme.

Um den Graphen, welcher aufgrund seiner vollständigen Darstellung der Dekomposition als präferierte Sicht angesehen werden kann, für möglichst große Ontologien sinnvoll einsetzen zu können, kann zum einen an dem Layout des Graphen gearbeitet werden. So könnten zum Beispiel Kanten, welche zum gleichen Knoten führen, zusammengeführt werden. Außerdem wäre es möglich, über Mechanismen nachzudenken, mit welchem sich Teile der Dekomposition verbergen bzw. anzeigen lassen.

Um mehr Informationen über den inhaltlichen Zusammenhang der Elemente zu erhalten, könnten die Techniken, welche im Kapitel zur signaturbasierten Modulextraktion eingeführt wurden, verwendet werden. Diese verfolgen das Ziel, den Inhalt möglichst übersichtlich und vollständig zu visualisieren. Dabei wäre es logisch, den dort als besten Ansatz bestimmten Ansatz zu wählen. Dieser könnte als eigene Ansicht in die Anwendung integriert werden. Des Weiteren bietet es sich an, in der Suchansicht die Elemente in einer eingerückten Liste zur Auswahl bereitzustellen.

Fazit

Insgesamt hat sich gezeigt, dass es mit der erweiterten atomaren Dekomposition möglich ist, den Prozess der Modulextraktion zu unterstützen. Die entwickelte Anwendung ist in der Lage, für kleine bis mittlere Ontologien den Nutzenden so durch den Prozess zu führen, dass dieser zu jedem Zeitpunkt weiß, was im gewählten Modul enthalten ist und die Auswirkungen von Ergänzungen ungefähr abschätzen kann.

Kapitel 7

Fazit und Ausblick

In dieser Arbeit sollte der Frage nachgegangen werden, wie Nutzenden bei der Spezifikation eines Moduls in einer Ontologie durch Visualisierungen unterstützt werden können.

Um diese Frage zu beantworten, sind unterschiedliche Ansätze untersucht worden. Dabei wurde grundsätzlich zwischen signaturbasierten Verfahren und dekompositionsbasierten Verfahren zur Modulextraktion unterschieden. Zunächst wurde untersucht, wie bestehende Visualisierungen von Ontologien im Kontext der Modulextraktion einsetzbar sind. Dabei wurde gezeigt, dass diese Visualisierungen die Signatur der Ontologie darstellen und damit für signaturbasierte Modulextraktion geeignet sind. Aus dem Vergleich der Visualisierungen ging hervor, dass die Kombination von geringer Komplexität und hohem Informationsgehalt ein Problem ist.

Für dekompositionsbasierte Verfahren hat sich die atomare Dekomposition als grundsätzlich geeignet herausgestellt. Daher wurde diese für Modulextraktion angepasst. Das Ergebnis dieser Modifikation wurde erweiterte atomare Dekomposition genannt und stellt die Dekomposition abhängig von einer Menge an gewählten Atomen dar. Außerdem wurde mit der Signaturerweiterung eine Beschriftung für die Atome eingeführt. Um dieses Verfahren zu visualisieren, sind drei Ansichten entwickelt worden, wobei die inkrementelle Ansicht und die Suchansicht die Probleme der Skalierbarkeit des Graphen umgehen.

Die entwickelte Anwendung auf Basis der erweiterten atomaren Dekomposition implementiert die drei Ansichten auf diese. In Fallbeispielen konnte anhand der Anwendung gezeigt werden, dass mit erweiterter atomarer Dekomposition Module aus kleinen bis mittleren Ontologien extrahiert werden können.

Ausblick

In diesem Abschnitt soll dargestellt werden, welche Fragestellungen in dieser Arbeit noch offen geblieben oder erst aus dieser entstanden sind. Zunächst hat diese Arbeit theoretische Grundlagen untersucht. Folglich wäre es sinnvoll, als Nächstes praktische Aspekte zu beleuchten. Spannend wäre es, das Interface für die Visualisierung der erweiterten atomaren Dekomposition genauer zu betrachten und weiterzuentwickeln.

Außerdem wäre neben dem theoretischen Vergleich der Ansätze in dieser Arbeit eine Nutzerstudie sinnvoll. Problematisch ist dabei jedoch, Umsetzungen für die einzelnen Ansätze auszuwählen. Zum Zeitpunkt der Arbeit ist bis auf die hier entwickelte Anwendung keine Anwendung bekannt, welche eine Visualisierung zur Unterstützung von Modulextraktion unterstützt. Für eine Nutzerstudie müssten daher Umsetzungen oder Prototypen für die einzelnen Ansätze entwickelt werden oder es müsste mit generellen Ontologievisualisierungen gearbeitet werden.

Wie bei der Evaluation der entwickelten Anwendung bereits erwähnt, scheint die Kombination von dekompositionsbasierten und signaturbasierten Verfahren vielversprechend zu sein. Um dies weiter zu untersuchen, könnte eine spezifische Umsetzung zu dieser Kombination erarbeitet werden.

Danksagung

Zunächst möchte ich mich bei Prof. Dr. Thomas Schneider für die Idee zum Thema der Arbeit, die Beantwortung etlicher Fragen und das detaillierte Feedback zu Ergebnissen bedanken. Außerdem bedanke ich mich bei Dr. Serge Autexier für das Interesse an dieser Arbeit und die Bereitschaft, diese zu prüfen. Schlussendlich bedanke ich mich bei Julia für ausführliches Korrekturlesen.

Kapitel 8

Bibliografie

[AL04] J. Angele und G. Lausen. “Ontologies in F-logic”. In: *Handbook on Ontologies*. Hrsg. von S. Staab und R. Studer. International Handbooks on Information Systems. Springer, 2004, S. 29–50.

[Baa+17] F. Baader, I. Horrocks, C. Lutz und U. Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.

[BHS04] F. Baader, I. Horrocks und U. Sattler. “Description Logics”. In: *Handbook on Ontologies*. Hrsg. von S. Staab und R. Studer. International Handbooks on Information Systems. Springer, 2004, S. 3–28.

[CMS99] S. K. Card, J. D. Mackinlay und B. Shneiderman. *Readings in information visualization - using vision to think*. Academic Press, 1999.

[Cue+09] B. Cuenca Grau, I. Horrocks, Y. Kazakov und U. Sattler. “Extracting Modules from Ontologies: A Logic-Based Approach”. In: *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*. Hrsg. von H. Stuckenschmidt, C. Parent und S. Spaccapietra. Bd. 5445. Lecture Notes in Computer Science. Springer, 2009, S. 159–186.

[Cue+08] B. Cuenca Grau, I. Horrocks, B. Motik, B. Parsia, P. F. Patel-Schneider und U. Sattler. “OWL 2: The next step for OWL”. In: *J. Web Semant.* 6.4 (2008), S. 309–322.

[Cue+06] B. Cuenca Grau, B. Parsia, E. Sirin und A. Kalyanpur. “Modularity and Web Ontologies”. In: *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*. Hrsg. von P. Doherty, J. Mylopoulos und C. A. Welty. AAAI Press, 2006, S. 198–209.

[Del+11a] C. Del Vescovo, D. Gessler, P. Klinov, B. Parsia, U. Sattler, T. Schneider und A. Winget. “Decomposition and Modular Structure of BioPortal Ontologies”. In: *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*. Hrsg. von L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal,

N. F. Noy und E. Blomqvist. Bd. 7031. Lecture Notes in Computer Science. Springer, 2011, S. 130–145.

[Del+19] C. Del Vescovo, M. Horridge, B. Parsia, U. Sattler, T. Schneider und H. Zhao. *Modular Structures and Atomic Decomposition in Ontologies*. Manuscript. *http://www.informatik.uni-bremen.de/~schneidt/dl2019/AD.pdf*. University of Bremen, 2019.

[Del+11b] C. Del Vescovo, B. Parsia, U. Sattler und T. Schneider. "The Modular Structure of an Ontology: Atomic Decomposition". In: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. Hrsg. von T. Walsh. IJCAI/AAAI, 2011, S. 2232–2237.

[DMW06] P. Doherty, J. Mylopoulos und C. A. Welty, Hrsg. *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*. AAAI Press, 2006.

[DTI07] P. Doran, V. A. M. Tamma und L. Iannone. "Ontology module extraction for ontology reuse: an ontology engineering perspective". In: *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management, CIKM 2007, Lisbon, Portugal, November 6-10, 2007*. Hrsg. von M. J. Silva, A. H. F. Laender, R. A. Baeza-Yates, D. L. McGuinness, B. Olstad, Ø. H. Olsen und A. O. Falcão. ACM, 2007, S. 61–70.

[FSH02] C. Fluit, M. Sabou und F. van Harmelen. "Ontology-based Information Visualization". In: *Visualizing the Semantic Web*. Hrsg. von V. Geroimenko und C. Chen. Springer, 2002, S. 36–48.

[Fre+02] C. M. D. S. Freitas, P. R. G. Luzzardi, R. A. Cava, M. A. Winckler, M. S. Pimenta und L. P. Nedel. "Evaluating Usability of Information Visualization Techniques". In: *Proceedings of the 5thSymposium on Human Factors in Computer Systems (IHC 2002)*. 2002, S. 40–51.

[GC02a] V. Geroimenko und C. Chen. "Preface". In: *Visualizing the Semantic Web*. Hrsg. von V. Geroimenko und C. Chen. Springer, 2002, S. vii–ix.

[GC02b] V. Geroimenko und C. Chen, Hrsg. *Visualizing the Semantic Web*. Springer, 2002.

[GLW06] S. Ghilardi, C. Lutz und F. Wolter. "Did I Damage My Ontology? A Case for Conservative Extensions in Description Logics". In: *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*. Hrsg. von P. Doherty, J. Mylopoulos und C. A. Welty. AAAI Press, 2006, S. 187–197.

[Gra+08] B. C. Grau, I. Horrocks, Y. Kazakov und U. Sattler. “Modular Reuse of Ontologies: Theory and Practice”. In: *J. Artif. Intell. Res.* 31 (2008), S. 273–318.

[Gru93] T. R. Gruber. “A translation approach to portable ontology specifications”. In: *Knowledge Acquisition* 5.2 (1993), S. 199–220.

[Hai+13] T. Haidegger, M. Barreto, P. Gonçalves, M. K. Habib, S. K. V. Ragavan, H. Li, A. Vaccarella, R. Perrone und E. Prestes. “Applied ontologies and standards for service robots”. In: *Robotics and Autonomous Systems* 61.11 (2013). Ubiquitous Robotics, S. 1215–1223.

[Kal+06] A. Kalyanpur, B. Parsia, E. Sirin, B. C. Grau und J. A. Hendler. “Swoop: A Web Ontology Editing Browser”. In: *J. Web Semant.* 4.2 (2006), S. 144–153.

[Kat+07] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis und E. G. Giannopoulou. “Ontology visualization methods - a survey”. In: *ACM Comput. Surv.* 39.4 (2007), S. 10.

[Kat+06] A. Katifori, E. Torou, C. Halatsis, G. Lepouras und C. Vassilakis. “A Comparative Study of Four Ontology Visualization Techniques in Protege: Experiment Setup and Preliminary Results”. In: *10th International Conference on Information Visualisation, IV 2006, 5-7 July 2006, London, UK*. IEEE Computer Society, 2006, S. 417–423.

[Kon+08] B. Konev, C. Lutz, D. Walther und F. Wolter. “Semantic Modularity and Module Extraction in Description Logics”. In: *ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings*. Hrsg. von M. Ghallab, C. D. Spyropoulos, N. Fakotakis und N. M. Avouris. Bd. 178. Frontiers in Artificial Intelligence and Applications. IOS Press, 2008, S. 55–59.

[Kut+04] O. Kutz, C. Lutz, F. Wolter und M. Zakharyaschev. “E-connections of abstract description systems”. In: *Artificial Intelligence* 156.1 (2004), S. 1–73.

[LNB14] S. Lohmann, S. Negru und D. Bold. “The ProtégéVOWL Plugin: Ontology Visualization for Everyone”. In: *The Semantic Web: ESWC 2014 Satellite Events*. Hrsg. von V. Presutti, E. Blomqvist, R. Troncy, H. Sack, I. Papadakis und A. Tordai. Cham: Springer International Publishing, 2014, S. 395–400.

[Mus14] M. A. Musen. “Chapter 3 - Knowledge Representation”. In: *Methods in Biomedical Informatics*. Hrsg. von I. N. Sarkar. Oxford: Academic Press, 2014, S. 49–79.

[Pan05] Z. Pan. “Benchmarking DL Reasoners Using Realistic Ontologies”. In: *Proceedings of the OWLED*05 Workshop on OWL: Experiences and Directions, Galway, Ireland, November 11-12, 2005*. Hrsg. von B. Cuenca Grau,

I. Horrocks, B. Parsia und P. F. Patel-Schneider. Bd. 188. CEUR Workshop Proceedings. CEUR-WS.org, 2005.

[PWG05] B. Parsia, T. Wang und J. Golbeck. "Visualizing Web Ontologies with CropCircles". In: *Proceedings of the ISWC 2005 Workshop on End User Semantic Web Interaction*. Hrsg. von A. Bernstein, I. Androutsopoulos, D. Degler und B. McBride. 2005.

[PJC09] J. Pathak, T. M. Johnson und C. G. Chute. "Survey of modular ontology techniques and their applications in the biomedical domain". In: *Integrated Computer-Aided Engineering* 16.3 (2009), S. 225–242.

[Pau19] R. A. de Paula. "Visualization Techniques". In: *Encyclopedia of Big Data Technologies*. Hrsg. von S. Sakr und A. Y. Zomaya. Cham: Springer International Publishing, 2019, S. 1775–1786.

[SSZ09] U. Sattler, T. Schneider und M. Zakharyaschev. "Which Kind of Module Should I Extract?" In: *Proceedings of the 22nd International Workshop on Description Logics (DL 2009), Oxford, UK, July 27-30, 2009*. Hrsg. von B. Cuenca Grau, I. Horrocks, B. Motik und U. Sattler. Bd. 477. CEUR Workshop Proceedings. CEUR-WS.org, 2009.

[Sch11] H. Schulz. "Treevis.net: A Tree Visualization Reference". In: *IEEE Computer Graphics and Applications* 31.6 (Nov. 2011), S. 11–15.

[SS04] S. Staab und R. Studer, Hrsg. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004.

[Sto+02] M.-A. Storey, N. F. Noy, M. Musen, C. Best, R. Fergerson und N. Ernst. "Jambalaya: An Interactive Environment for Exploring Ontologies". In: *Proceedings of the 7th International Conference on Intelligent User Interfaces*. IUI '02. San Francisco, California, USA: ACM, 2002, S. 239–239.

[Stu11] H. Stuckenschmidt. "Symbole, Objekte und Konzepte". In: *Ontologien: Konzepte, Technologien und Anwendungen*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 5–24.

Anhang A

Betrachtete Tools

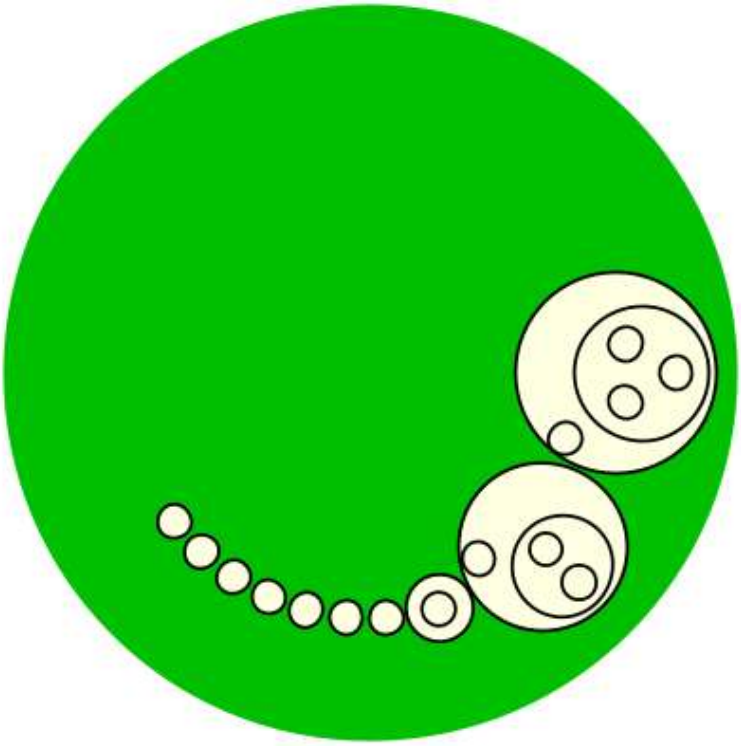

Abbildung A.1 CropCircles in Swoop(Koala-Ontologie)

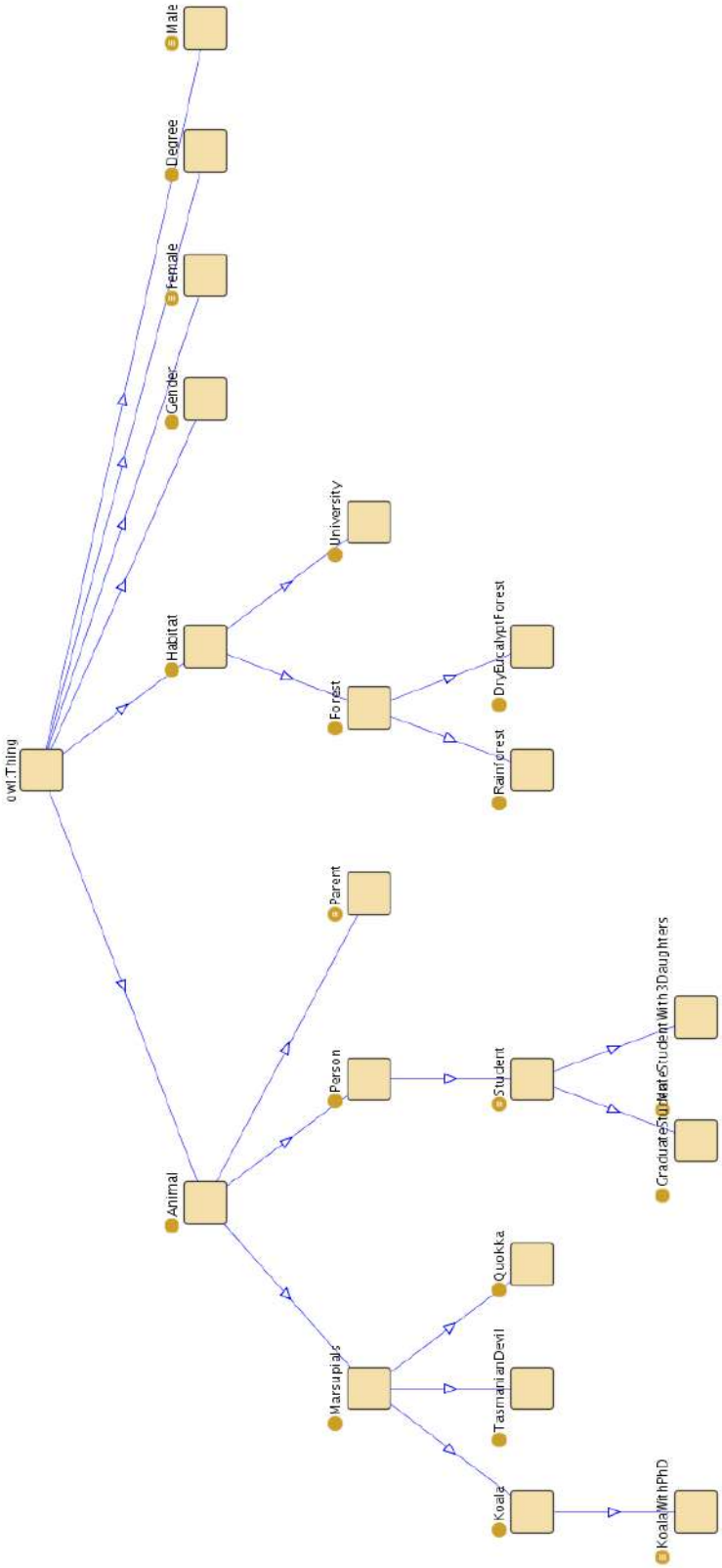

Abbildung A.2 Beispiel eines Baumes mit expliziten Kanten aus Jambalaya

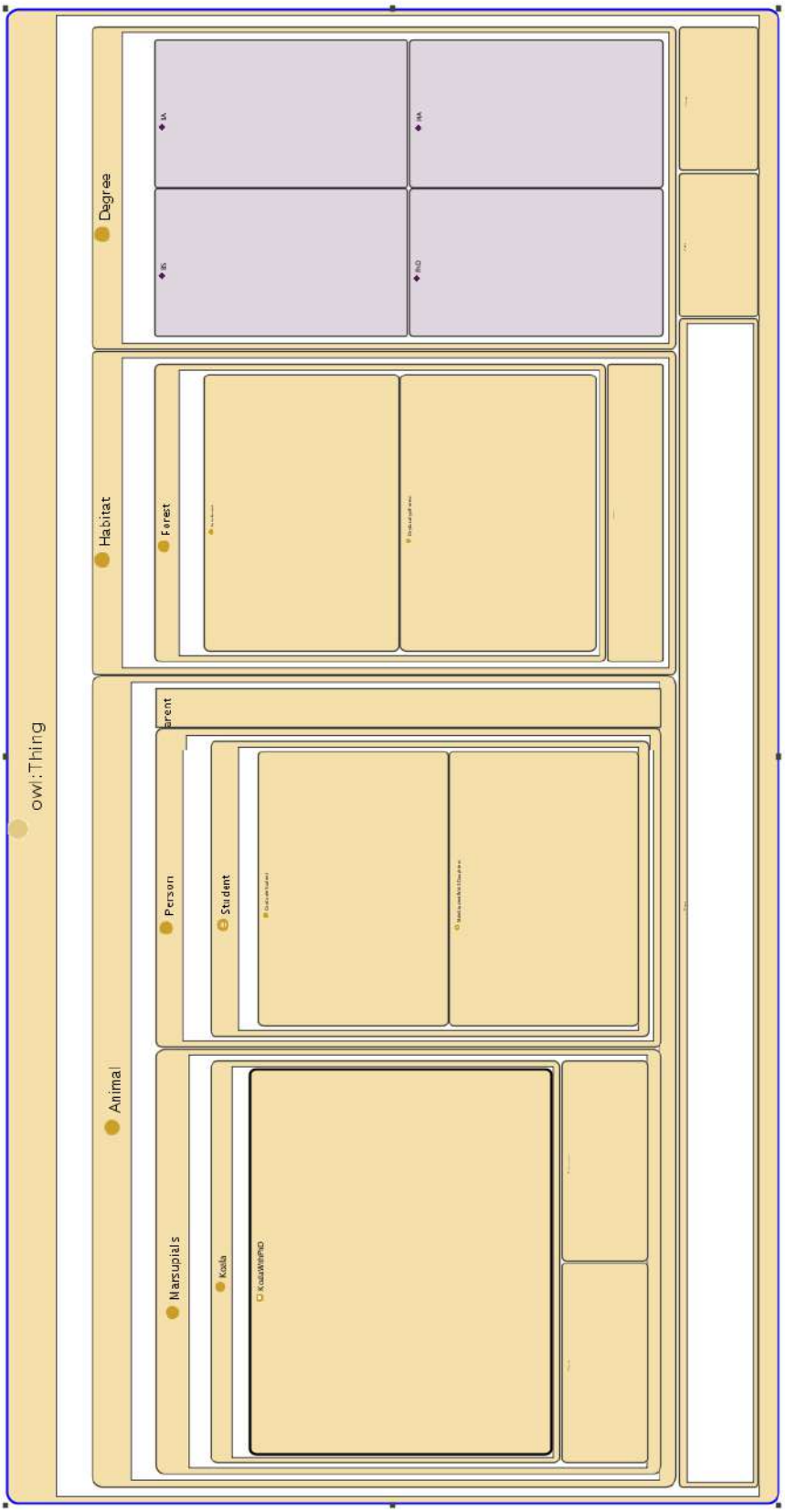

Abbildung A.3 Treemap-Ansicht in Jambalaya(Koala-Ontologie)

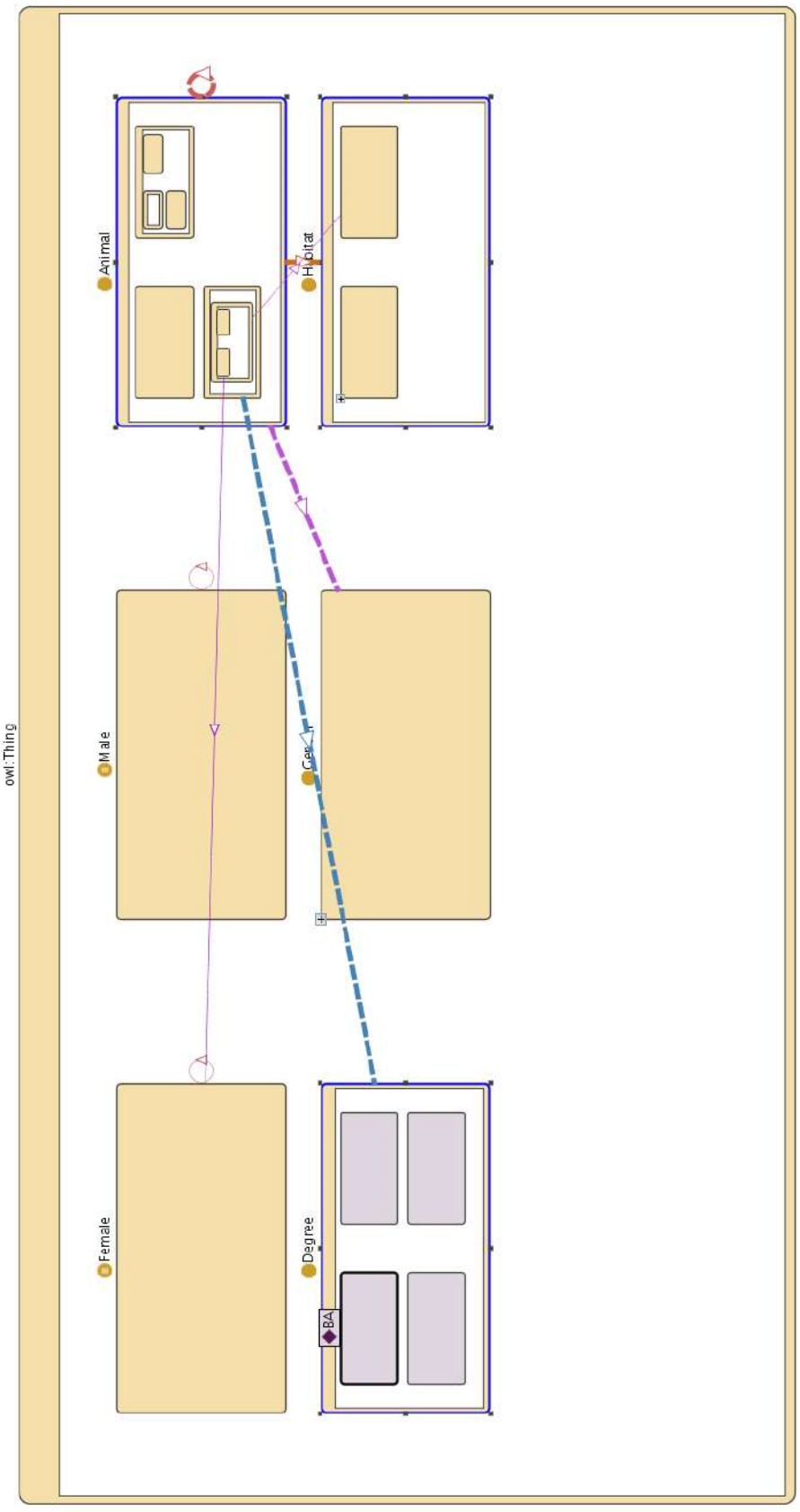

Abbildung A.4 „Nested View“ aus Jambalaya (Koala-Ontologie)

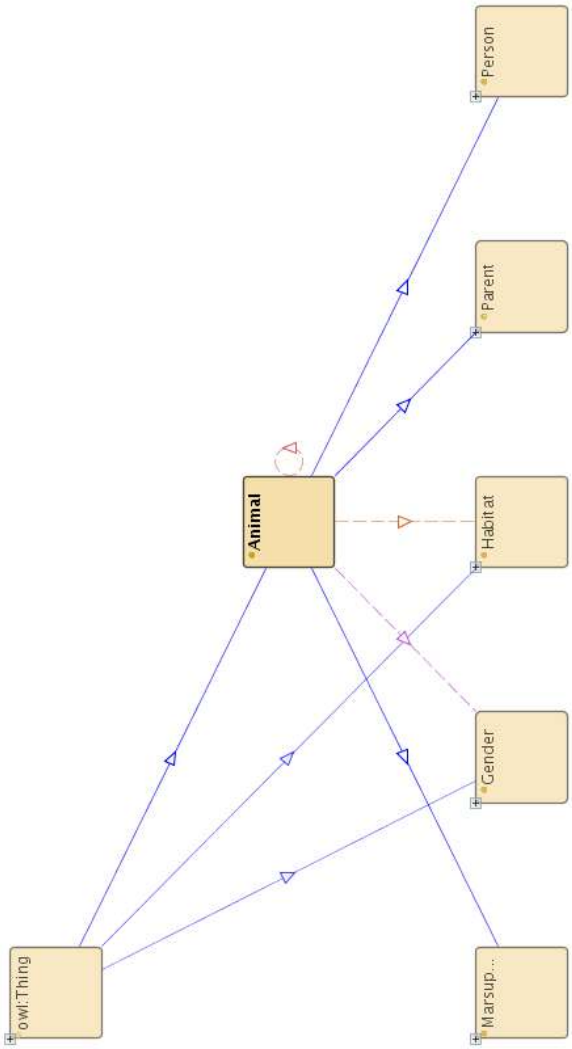

Abbildung A.5 „Query View“ aus Jambalaya (Koala-Ontolgie)

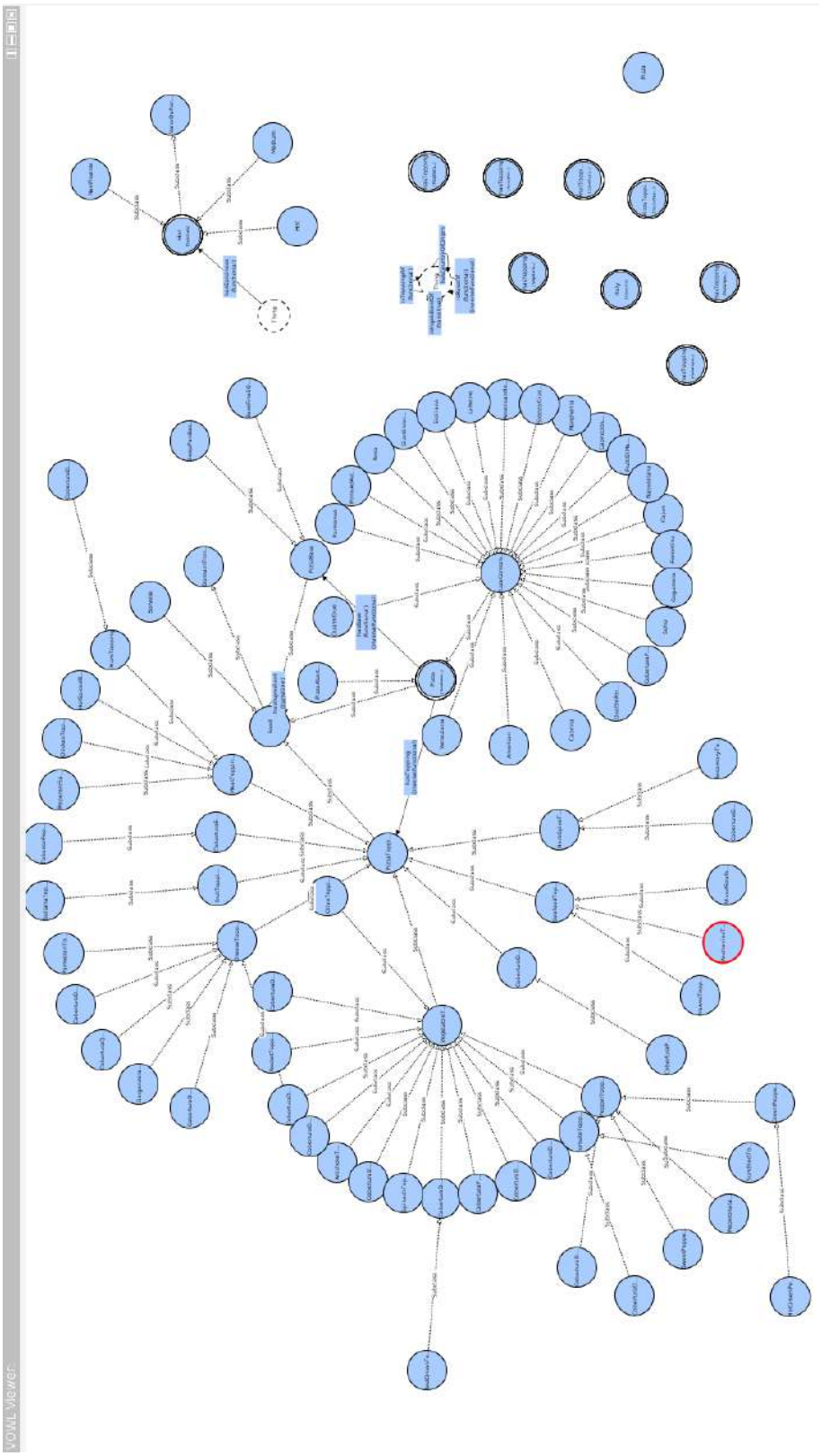

Abbildung A.6 VOWL Visualisierung der Pizza-Ontologie

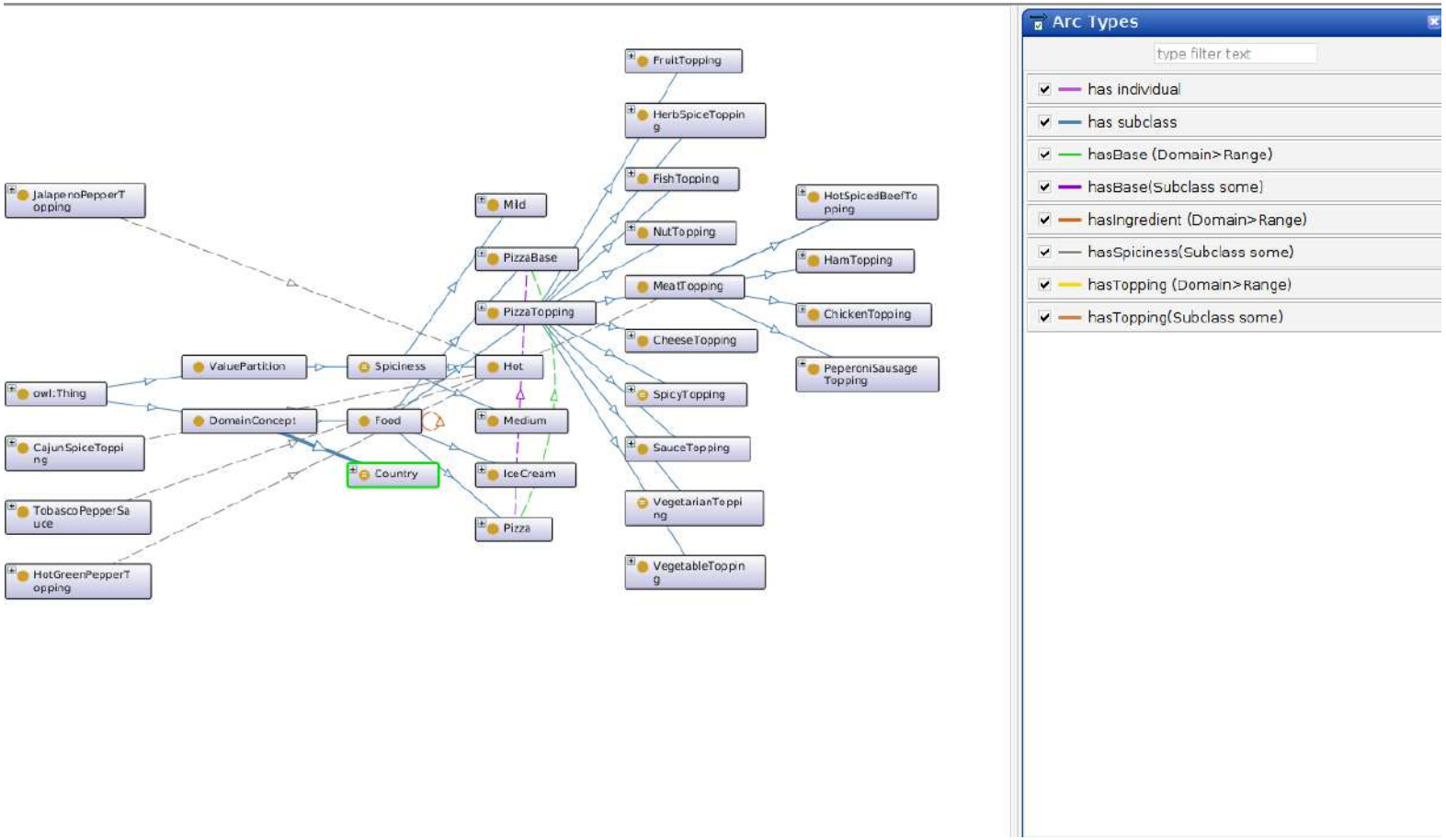

Abbildung A.7 Ontograf Viesualisierung der Pizza-Ontologie

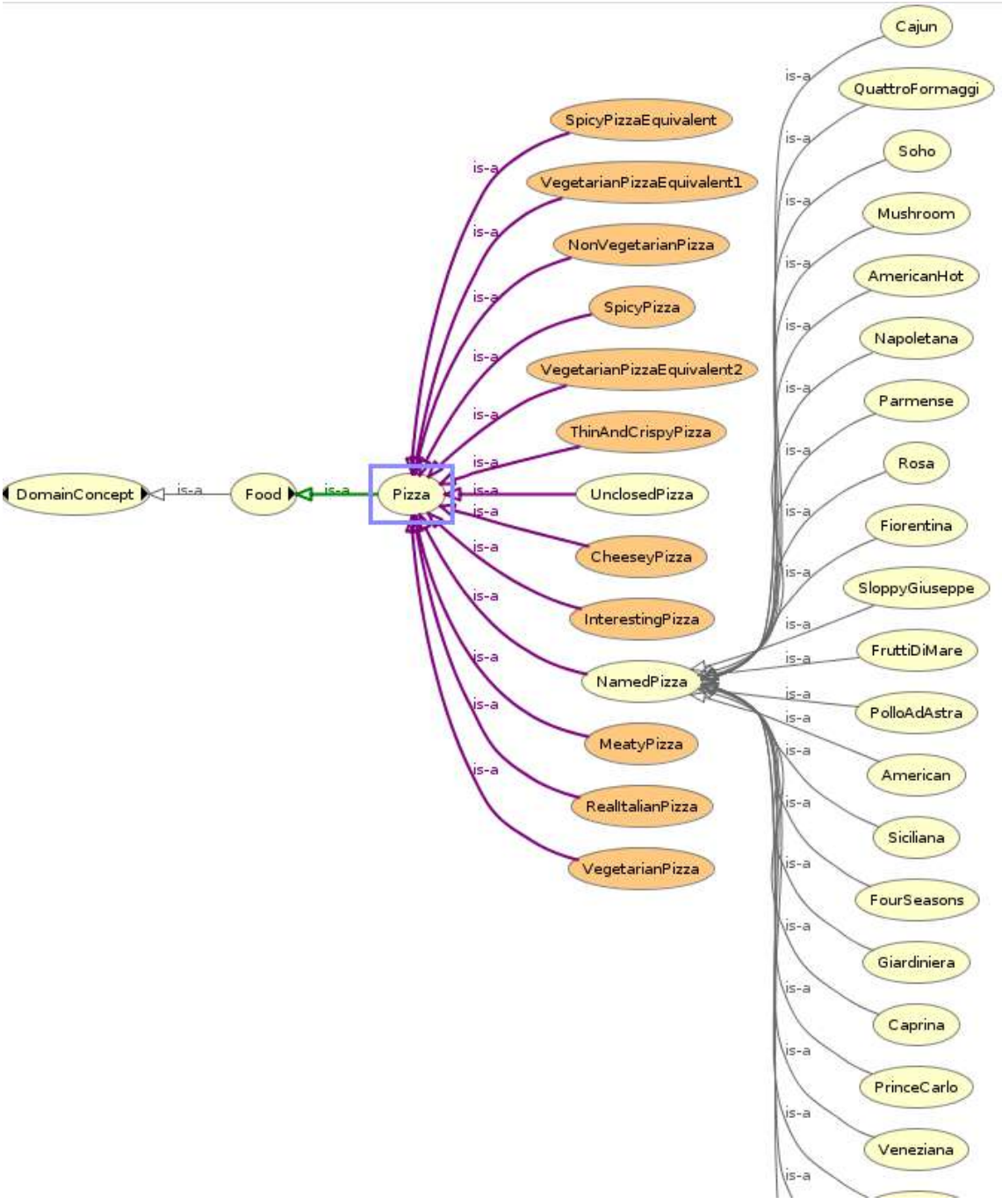

Abbildung A.8 Baum in OWLViz(Teil der Pizza-Ontologie)

Anhang B

Implementation

B.1 Ansichten

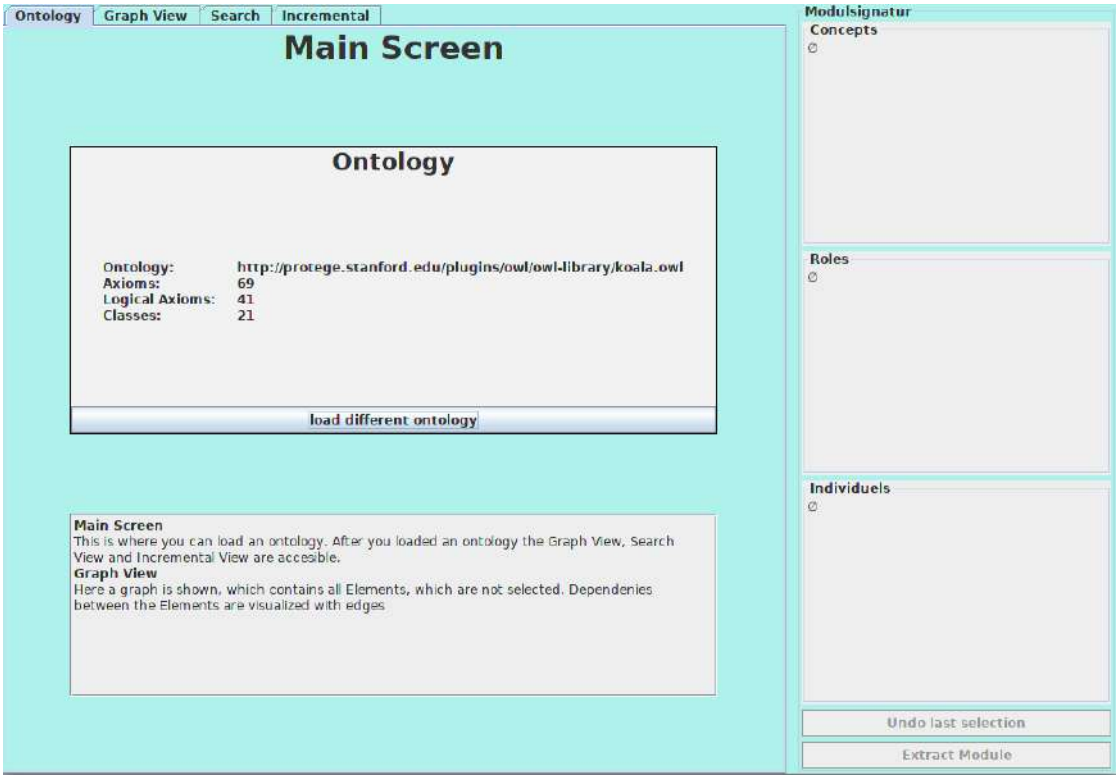

Abbildung B.1

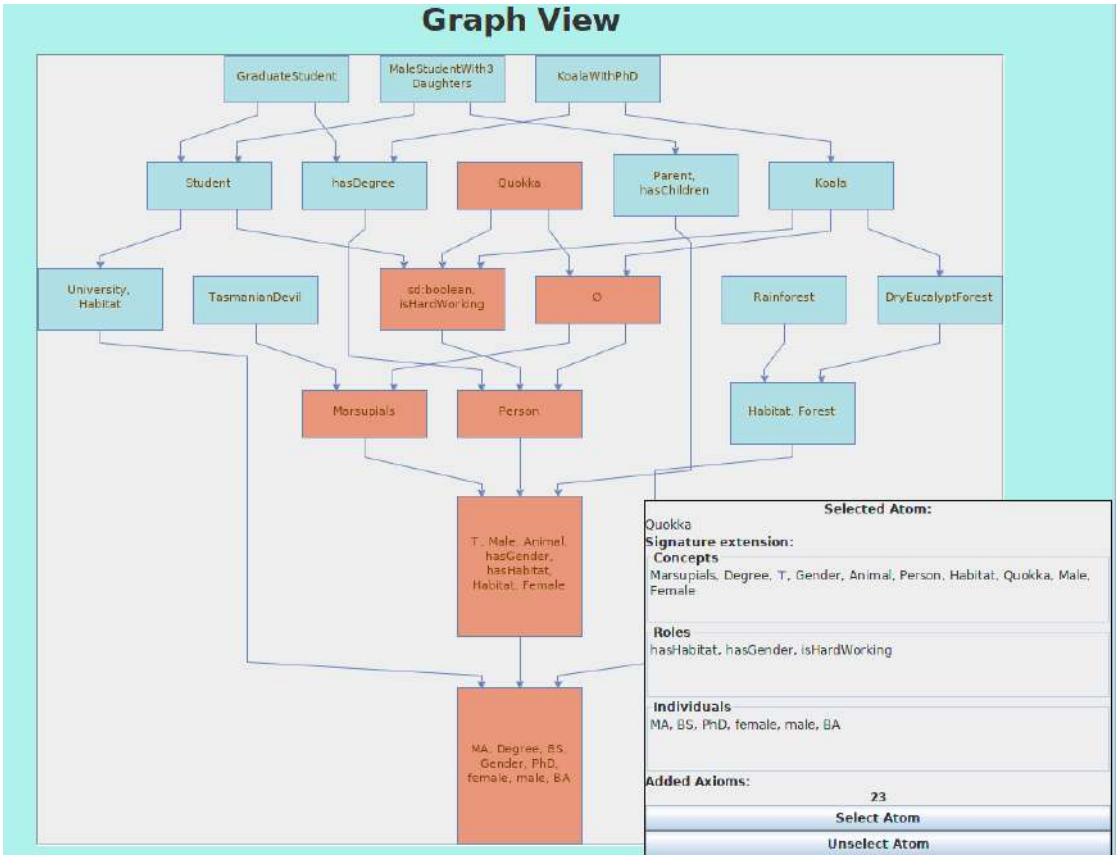

Abbildung B.2

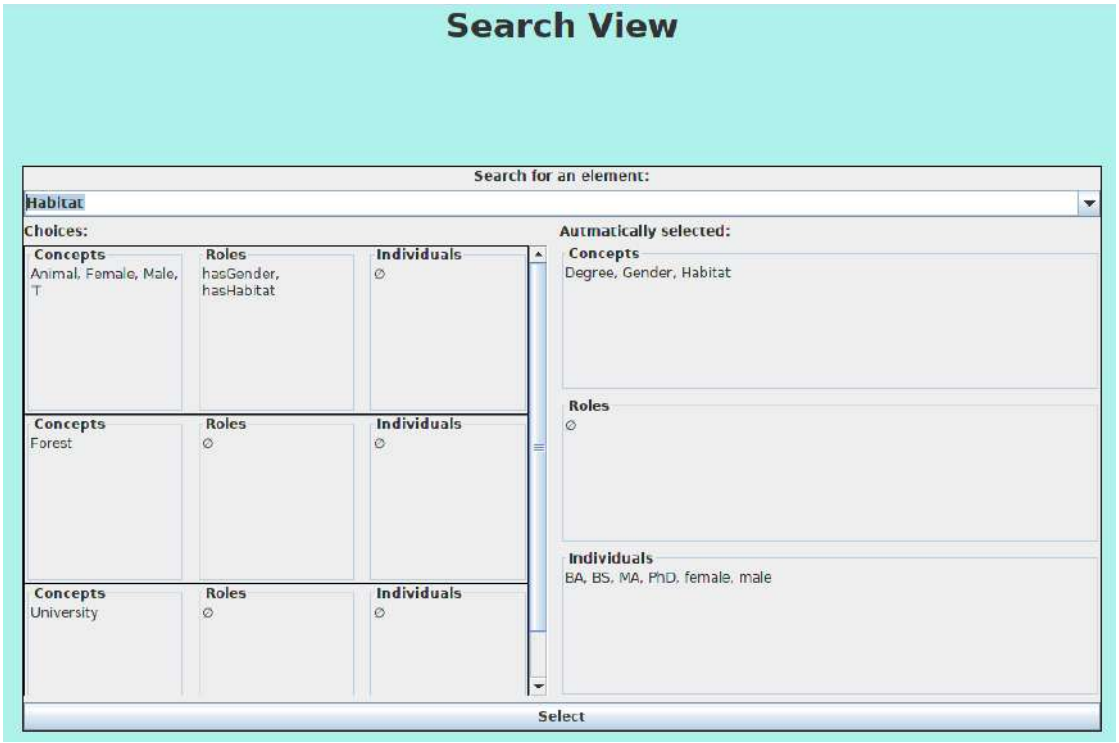

Abbildung B.3

Abbildung B.4

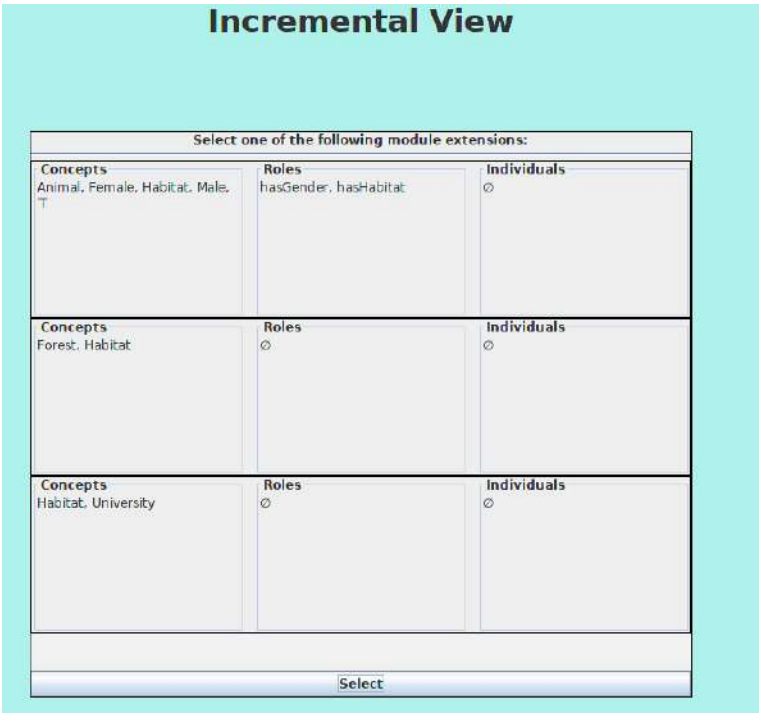

Abbildung B.5

B.2 Kurzbeschreibung der Klassen

App Startet die Anwendung.

atomPanel Anzeige eines ausgewählten Atoms für die Graphansicht.

AtomRenderer Stellt eine Repräsentation eines Atoms als Listenelement zur Verfügung.

CurrentState Verwaltet die geladene Ontologie, die aktuelle erweiterte atomare Dekomposition und die gewählten Axiome.

eadView Interface für Objekte, welche bei einer Veränderung in der erweiterten atomaren Dekomposition aktualisiert werden müssen.

extractScreen Ermöglicht das Extrahieren eines gewählten Moduls.

graphView Graphansicht auf die erweiterte atomare Dekomposition.

incrementalSelection Inkrementelle Ansicht auf die erweiterte atomare Dekomposition.

LayeredPaneLayout und StupidCenterLayout Werden für die Darstellung des Graphen benötigt, sind aber inhaltlich uninteressant.

MainScreen Hauptmenü, in welches die einzelnen Ansichten und die Anzeige des aktuellen Moduls eingebettet sind.

ontologySceen Anzeige zum Verwalten der verwendeten Ontologie. Vor allem für das Laden der Ontologie bedeutsam.

OWLEnitityRenderer Ist in der Lage, OWLEntities sinnvoll als Zeichenkette auszugeben.

OWLEntityWrapper Beinhaltet eine OWLEntity und einen entsprechenden Renderer, um die Entity darstellen zu können.

searchLayout Suchansicht auf die erweiterte atomare Dekomposition.

SignatureUtils Hilfsfunktionen für den Umgang mit Signaturen. Dazu gehört das Erstellen, Filtern und als Zeichenkette Ausgeben dieser.

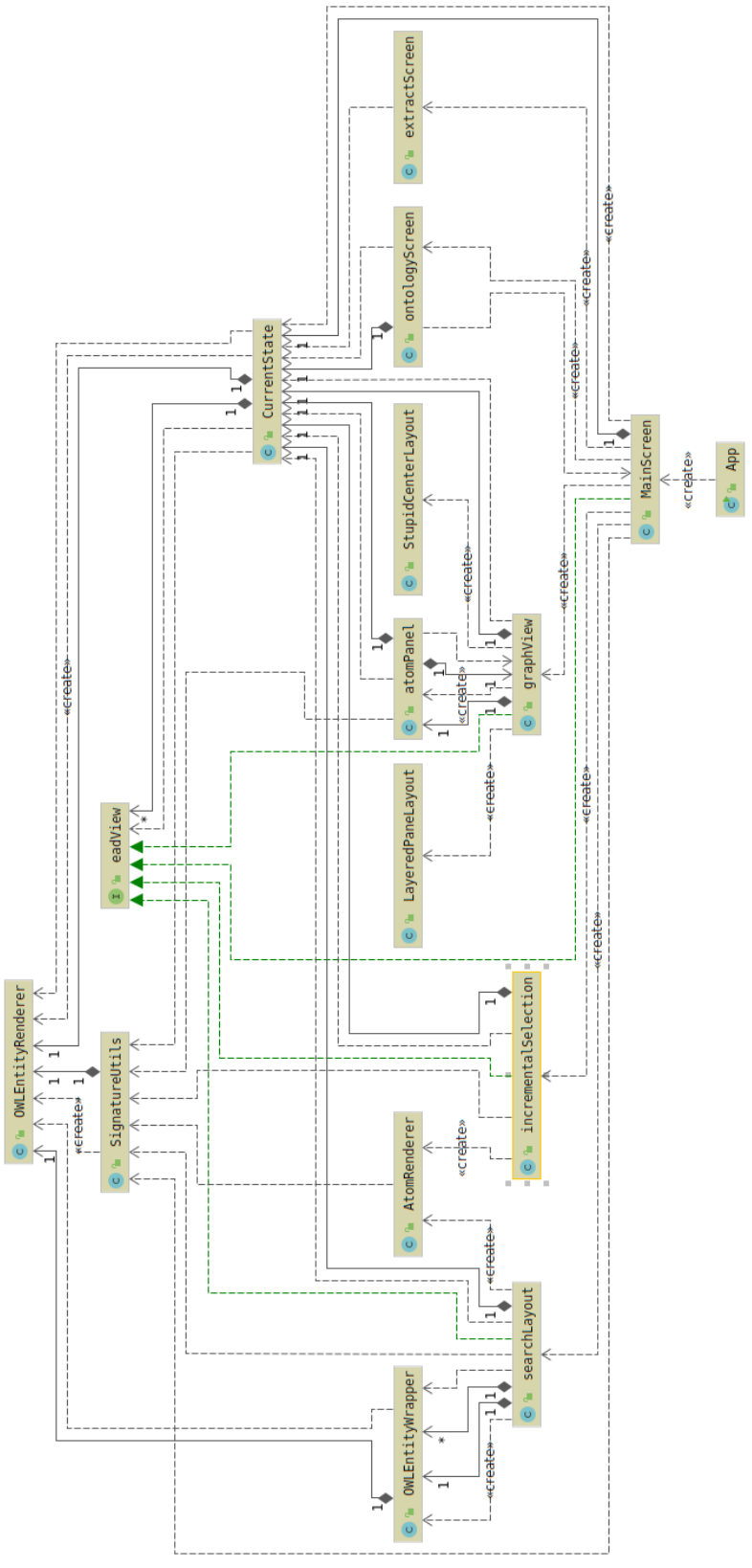

Abbildung B.6 UML Diagramm der Implementation

Anhang C

Digitale Abgabe

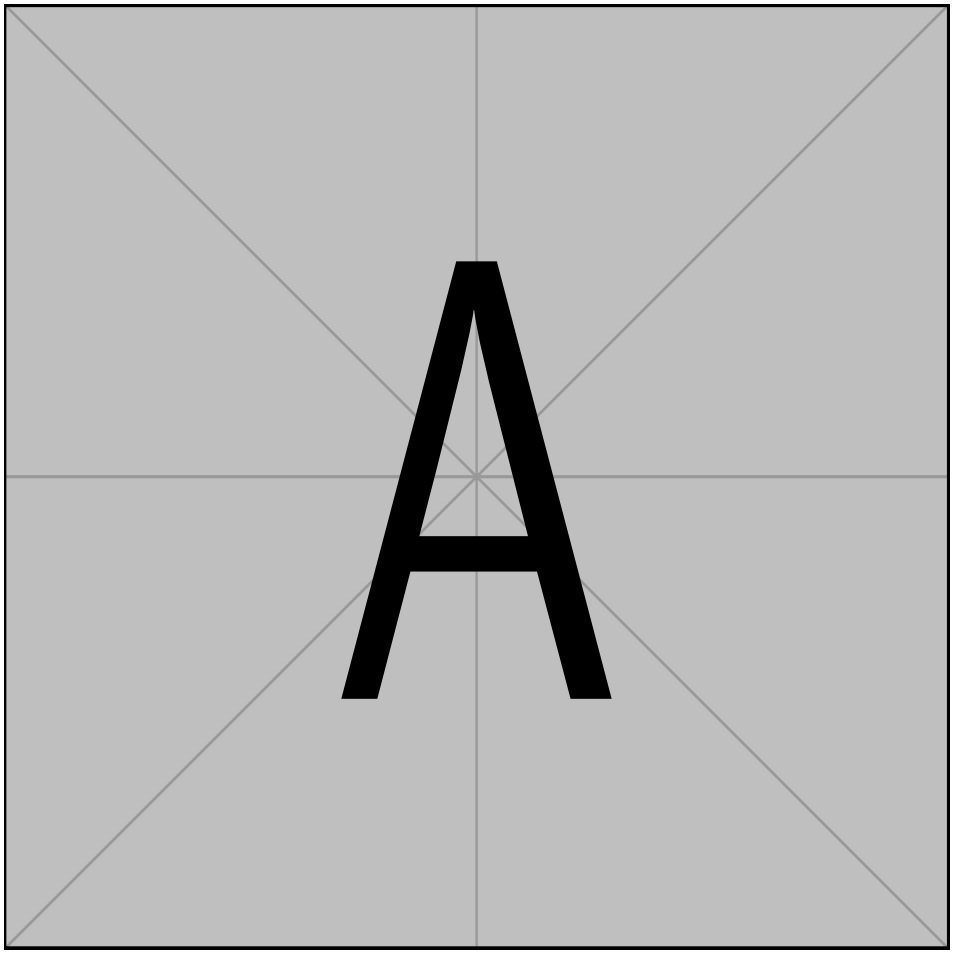

Abbildung C.1 Die physikalisch abgegebene Version der Arbeit enthält an dieser Stelle eine CD mit der digitalen Version der Arbeit, sowie der in Rahmen dieser Arbeit erstellten Implementation.