Karlsruhe Institute of Technology

Programmieren in natürlicher Sprache: Aufbau einer Alice-Ontologie – Korpus-Ontologie-Assoziation –

Studienarbeit
von

Sebastian Weigelt

An der Fakultät für Informatik
Institut für Programmstrukturen
und Datenorganisation (IPD)

Erstgutachter: Prof. Dr. Walter F. Tichy
Betreuender Mitarbeiter: Dipl. Inform.-Wirt Mathias Landhäußer

Bearbeitungszeit: 08. August 2012 – 07. November 2012

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, 07.11.2012

.....

(**Sebastian Weigelt**)

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

Informatikern fällt es leicht mit Rechnern zu kommunizieren. Durch den Einsatz unterschiedlicher Programmiersprachen ist es ihnen möglich, dem Rechner explizite Anweisungen zu geben und gewünschte Berechnungen durchzuführen. Für fachfremde Personen stellt das Erlernen einer Programmiersprache hingegen meist einen mit dem Ergebnis nicht in Relation stehenden Aufwand dar. Viele nutzen zwar Personal Computer, Laptop, Smartphone, Tablet oder Ähnliches, doch nur die wenigsten Nutzer können tatsächlich programmieren. Folglich wäre es wünschenswert, das Programmieren als schöpferische Tätigkeit zu vereinfachen und somit für mehr Menschen zugänglich zu machen. Ein potentieller Ansatz, um dies zu bewerkstelligen, ist Programmieren in natürlicher Sprache zu ermöglichen. Für ein Computer-Programm müsste dann kein Quelltext in einer entsprechenden Programmiersprache erarbeitet werden. Vielmehr würden natürlichsprachliche Befehle und Beschreibungen ausreichen. Programmieren wird so intuitiver, leichter erlernbar und vor allem kreativer. Ein erster Schritt zur Verwirklichung dieser Vision soll im Projekt „Programmieren in natürlicher Sprache“ getan werden.

1.1. Das Gesamtprojekt

Die Idee, natürlichsprachliche Beschreibungen zum Programmieren zu verwenden, ist nicht neu. Der Ansatzpunkt, der für das Projekt „Programmieren in natürlicher Sprache“ gewählt wurde, unterscheidet es jedoch von vorangegangen Bemühungen. Nahezu alle bisherigen Versuche (siehe Kapitel 3.1) verfolgten mehr oder weniger die Grundidee, den natürlichsprachlichen Text zu analysieren und daraus Konzepte zu bilden, die sich später auf eine gewählte Programmiersprache übertragen lassen. Der hiesige Entwurf verfährt entgegengesetzt; Konzepte einer Programmiersprache werden modelliert bevor versucht wird, den natürlichsprachlichen Text auf diese Modellierung abzubilden. Hierfür wird als Rahmenwerk *Alice* [Con97] [CAB+00] gewählt. *Alice* ist eine einfache, stark objektorientierte Programmiersprache, die vor allem für Programmieranfänger geeignet ist. Mit *Alice* können per Drag-And-Drop 3D-Welten programmiert oder sogar kleine Videos animiert und Spiele entwickelt werden. Gleichzeitig bietet *Alice* aber auch klassische Programmierkonzepte, wie Variablen, Methoden, Schleifen und Ähnliches. Da in *Alice* beinahe die gesamte Funktionalität durch bereits vorhandene Objekte und Methoden abgedeckt wird, lässt sich das Verhalten beziehungsweise der möglich Umgang mit *Alice* besonders gut modellieren. Diese Modellierbarkeit bildet den Ansatzpunkt für das hiesige Projekt.

Im Rahmen des Projektes soll ein System entstehen, das natürlichsprachliche Begriffe

mit Konzepten aus *Alice* verbindet. Ziel des Gesamt-Projektes ist es, aus einem Text automatisch ein *Alice*-Skript/Video zu erstellen.

Abbildung 1.1.: Schematische Darstellung der Arbeitsschritte

Wie in Abbildung 1.1 zu sehen, wird zunächst aus einem zuvor kreierten Video ein beschreibender Text (beziehungsweise mehrere Texte) erstellt. So entsteht ein Textkorpus. Dieser wird dann von dem im Projekt erstellten System analysiert und infolgedessen, mithilfe der angelegten Ontologie ein *Alice*-Skript erzeugt, welches wiederum in ein *Alice*-Video umgesetzt wird. Im Idealfall ist das neu generierte Video dem ursprünglichen sehr ähnlich. Dies wäre ein deutlicher Indikator für die erfolgreiche Umsetzung von natürlicher Sprache in Quelltext.

1.2. Lexikographische Untersuchung von *Alice*-Bestandteilen

Der in dieser Arbeit für das Gesamtprojekt geleistete Beitrag kann zwischen den Schritten drei und vier des Schemas in Abbildung 1.1 angesiedelt werden. Zuvor wurden im Rahmen der Bachelorarbeit von Oleg Peters [Pet12] die Programmiersprache *Alice* und die vorhandenen Objekte, sowie deren Methoden und Eigenschaften mittels einer Ontologie modelliert. In der hier vorgestellten Arbeit soll nun der Übergang vom Textkorpus zum bestehenden Modell (Ontologie) erleichtert werden.

Eine intuitive Möglichkeit, dies zu bewerkstelligen, wäre, eine zweite Ontologie aus dem Text-Korpus zu erstellen und diese mit der vorhanden zu verbinden. Jenes Vorgehen wird *Ontology-Matching/Mapping* genannt und zum Beispiel von Noy beschrieben [Noy09]. Ziel hierbei ist, zwei Ontologien zu vergleichen und zu einer einzigen zu verschmelzen, die entweder nur aus der Schnittmenge oder aber der Additionsmenge der ursprünglichen Ontologien besteht. Hierfür gibt es unterschiedliche Ansätze. Einer dieser Ansätze, von Giunchiglia et. al. [GS03] [GSY04] nutzt unter anderem ein Synonym-Wörterbuch, um Kongruenzen zwischen den Ontologien zu ermitteln. Da *Ontologie-Matching/Mapping* für den hiesigen Kontext einen unangemessen Mehraufwand darstellt, der zudem nur bedingt zielführend wäre, soll stattdessen die Idee der Ermittelung von Synonymen näher betrachtet werden.

Dabei wird angenommen, dass Objekte im Modell Namen tragen, die bereits Auskunft über ihrer Verwendung in *Alice* geben. Das Modell (Ontologie) von *Alice* soll derart erweitert werden, dass später zu nutzende Objekt-, Methoden- und Parameter-Namen lexikographisch untersucht und Synonyme bestimmt werden. Diese Vorarbeit erleichtert die spätere Verknüpfung von Textkorpus und Modell (Ontologie) erheblich, da natürlichsprachliche Beschreibungen in den meisten Fällen nicht exakt der *Alice*-Syntax entsprechen; es soll also versucht werden, Variation in natürlicher Sprache mittels Einflechtung von Synonymen in das Modell auszugleichen.

Das grundlegende Vorgehen ist dabei wie folgt: Ein Objekt beziehungsweise dessen Name, der zumeist aus einer kurzen Wortgruppe besteht, wird aus dem Modell extrahiert. Das sinngebende Wort der Wortgruppe wird ermittelt und Synonyme für selbiges bestimmt.

Nicht treffende Synonyme werden aussortiert, bevor abschließend die Synonyme in geeigneter Weise in die bestehende Ontologie eingegliedert werden.

Im Rahmen dieser Zielsetzung stellen sich diverse Herausforderungen. So muss zunächst geklärt werden, welches das sinngebende Wort der Wortgruppe ist beziehungsweise ob ein solches überhaupt existiert. Gleichfalls ist die Entscheidung, ob ein Synonym für den vorliegenden Kontext relevant ist, keineswegs trivial. Zuletzt stellt sich die Frage, wie Synonyme in die Ontologie eingebettet werden sollten; die Informationen müssen gut zugänglich und flexibel gespeichert werden.

Kapitel 2 liefert einige Grundlagen zum besseren Verständnis der Thematik, bevor Kapitel 3 verwandte Arbeiten, sowohl bezüglich des Gesamt-Projektes als auch der hier geleisteten Arbeit, aufgreift und vergleicht. Kapitel 4 beschreibt die grundlegenden Ideen hinter dem hier gewählten Ansatz und erläutert den Entwurf, dessen Details in der Implementierung in Kapitel 5 erläutert werden. Die geleistet Arbeit wird anschließend in Kapitel 6 evaluiert und die Ergebnisse in Kapitel 7 zusammengefasst.

2. Grundlagen

Die Umsetzung des Projekts erfordert den Einsatz unterschiedlichster Techniken zur Wissensaufbereitung und -darstellung sowie der Sprachverarbeitung. Zur Wissensrepräsentation wurde für das Projekt eine Ontologie gewählt, während zur Bestimmung der Wortarten in den Objekt- und Methodennamen ein Wortart-Markierer (Part-Of-Speech-Tagger) zum Einsatz kommt. Schließlich soll eine Art Synonym-Datenbank die sprachliche Erweiterung der Ontologie ermöglichen. Die eingesetzten Technologien sollen im Folgenden kurz erläutert werden.

Zunächst wird jedoch das gewählte Rahmenwerk für dieses Projekt, die Programmiersprache *Alice* [Con97] [CAB+00], einer genaueren Betrachtung unterzogen. Über eine graphische Oberfläche können per Drag-And-Drop kleine Programme erstellt werden, meist in Form von Videos oder Mini-Spielen. Da das Ziel von *Alice* darin besteht, Programmieranfängern den Einstieg in das objektorientierte Programmieren zu ermöglichen, ist auch *Alice* als Sprache stark objektorientiert. Gleichzeitig wird von *Alice*, um Drag-And-Drop zu ermöglichen, eine Vielzahl von Methoden, Objekten und Attributen angeboten. Diese Eigenschaften machen *Alice* leicht überschau- und beherrschbar, wobei die Objektorientierung und die angebotenen Bibliotheksobjekte und -Methoden den optimalen Ansatzpunkt für unser Versuchsumfeld bieten. Im Rahmen des Projekts „Programmieren in natürlicher Sprache“ soll die Konzentration zunächst auf der Umsetzung der *Alice*-API in natürliche Sprache liegen. Abbildung 2.1 zeigt die graphische Oberfläche von *Alice* und ein kurzes Beispiel-Skript[1]. Gleichzeitig ergibt sich die Nutzung einer objektorientierten Programmiersprache für dieses Projekt ganz logisch, um die vorgegebenen Eigenschaften des ebenfalls objektorientierten Rahmenwerks optimal erhalten und verwenden zu können. In dieser Studienarbeit wird aus diesen Gründen als Programmiersprache Java verwendet.

Die Herausforderung dieses Projekts besteht unter anderem darin, die API der gewählten Programmiersprache, hier *Alice*, geeignet in einer Art Wissensdatenbank darzustellen. Die starke Objektorientierung von *Alice* mit den üblichen Programmierkonzepten, wie Vererbung und Abstraktion, legt die Verwendung einer hierarchischen Wissensrepräsentation nahe; eben dies bieten Ontologien. Guarino et. al [GOS09] definieren Ontologien als eine spezielle Form eines Informationsobjekt im Sinne der elektronischen Datenverarbeitung, welches die Struktur eines Systems formal modelliert. Die Systeme, die hierbei modelliert werden, sind von beliebiger Herkunft. Ein anschauliches Beispiel wäre ein System,

[1] Das Skript gehört zu der in den Kapiteln 4 und 5 beschriebenen Beispiel-*Alice*-Welt. Die zugehörige Ontologie in Kapitel 6 ist $O(W_1)$.

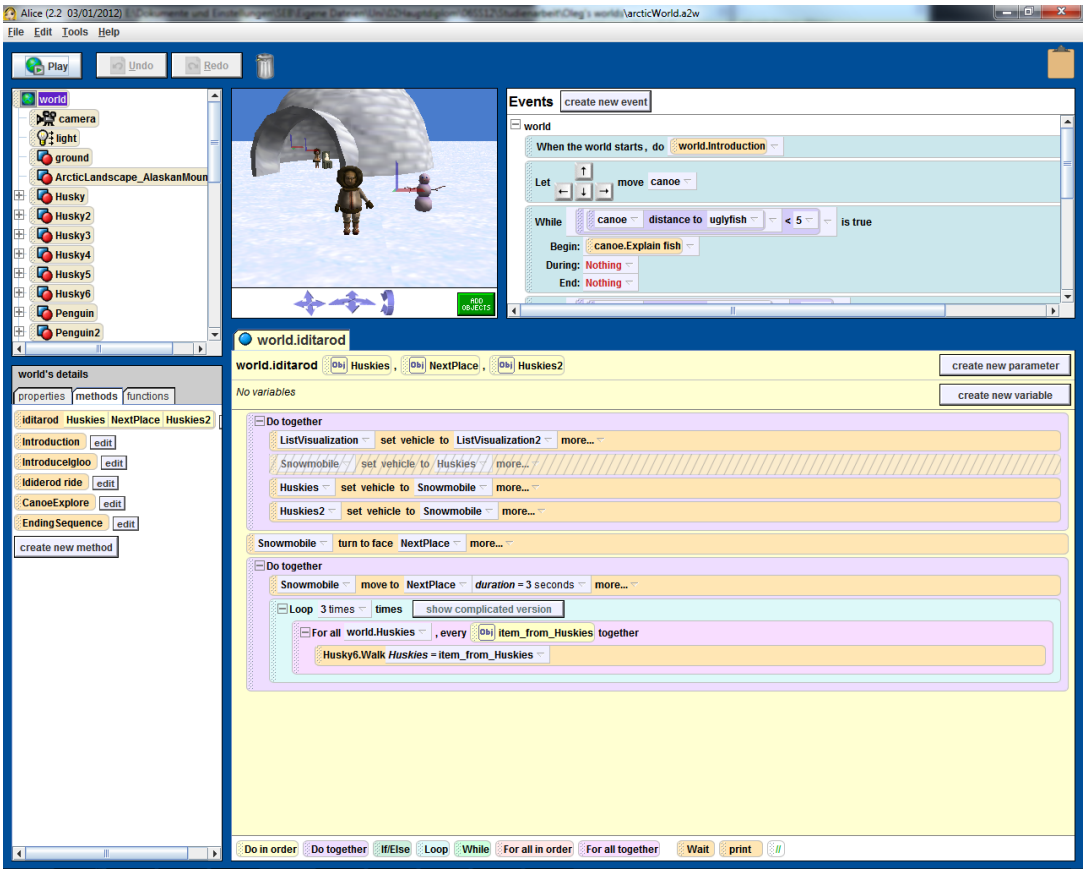

Abbildung 2.1.: Graphische Benutzeroberfläche von *Alice*

bestehend aus einem Unternehmen, dessen Angestellten und deren Beziehung zueinander. Entscheidend für die Darstellung in einer Ontologie, sind mögliche Generalisierungen/Spezialisierungen innerhalb des Systems, denn die hierarchische Beschreibung von Systemen ist das Haupteinsatzgebiet von Ontologien. Hierzu bieten Ontologien im wesentlichen drei Konzepte. Zunächst können Individuen als sogenannte Entitäten angelegt werden. Können diese unter einer gemeinsamen Idee zusammengefasst werden, bilden sie ein Konzept, auch Klasse genannt, der Ontologie. Entitäten können hierbei grundsätzlich auch unterschiedlichen Konzepten angehören. Ebenso können Konzepte zu einem übergeordneten Konzept zusammengefasst werden. Zuletzt können Relationen angelegt werden. Diese können zwischen Konzepten, Entitäten oder auch zwischen einem Konzept und einer Entität bestehen. Man unterscheidet zwischen uni- und bidirektionalen Relationen. Im eingangs gewählten Beispiel, wären unter anderem die einzelnen Mitarbeiter des Unternehmens Entitäten, die gemeinsam Konzepte wie „Verwaltungsangestellte“, „Manager“, „Wissenschaftler“ usw. bilden, wobei Relationen wie „ist Chef von“ (unidirektional) oder „arbeitet in Arbeitsgruppe mit“ (bidirektional) möglich wären. Dieses Beispiel ist ebenfalls in Abbildung 2.2 dargestellt. Ein weiteres Beispiel im Programmierumfeld wäre die Instanziierung (Entität) von abstrakten Klassen (Konzept) in objektorientierten Programmiersprachen wie Java.

Die hier vorgestellte Arbeit nutzt sowohl als Grundlage als auch zur Bearbeitung eine Ontologie, die von Oleg Peters zur Modellierung der *Alice*-API erstellt wurde. Die Ontologie wurde in OWL [Hor08] angelegt, die derzeit weitverbreitetste und höchstentwickelte Ontologie-Sprache. OWL bietet in der aktuellen Version OWL 2 neben den Standard-Ontologie-Konzepten wie Individuen (Entitäten), Klassen (Konzepten) und Eigenschaften (Relationen) auch einige Erweiterungen: Neben weitreichenden Optionen zum logischen Folgern gibt es die Möglichkeit, für Individuen oder Klassen Relationen anzulegen, die mit

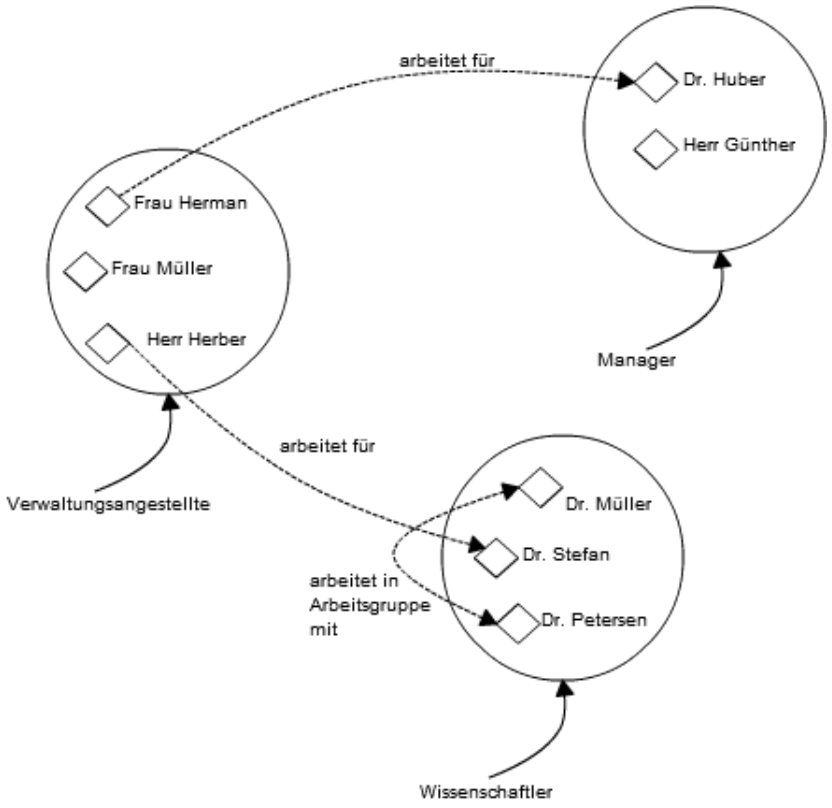

Abbildung 2.2.: Ontologie-Beispiel: Unternehmen

Datentypen (z.B. Integer) verknüpft sind. Weiterhin können Restriktionen erstellt werden, die z.B. eine Klasse als disjunkt zu einer anderen kennzeichnen oder Gleichheit zwischen Individuen herstellen. Für die vorliegende Arbeit ist jedoch eine andere Option essentiell: Zu jeder Klasse, jedem Individuum und jeder Relation können Annotationen erstellt werden. Jeder Annotation wird hierbei ein zusätzliches Markierungsfeld angefügt. Dieses Feld ist ursprünglich dafür vorgesehen, für die Annotation anzugeben, in welcher Sprache diese verfasst wurde. Gleichzeitig können die Annotationen über die Markierungsfelder unterschieden werden. Folglich können mehrere Kommentare an ein Ontologie-Objekt angehängt werden. Siehe hierzu auch Kapitel 5.8, sowie die Abbildungen 5.4 und 5.5.

Die *Alice*-API ist größtenteils, wie üblich, mit sprechenden Funktionsnamen versehen, das heißt, der Name der Funktion lässt auf die Aufgabe selbiger schließen. Ebenso haben einige *Alice*-API-Objekte, die im späteren Programm als Methode verwendet werden können, ein weiteres Textfeld, welches die Methode beschreibt. Wie Funktionsname und Textfeld für die Synonym-Findung verwendet werden, wird in Kapitel 4 genauer beschrieben; es ist jedoch unter anderem nötig, die Wortarten der Einzelwörter im Textfeld beziehungsweise Funktionsnamen zu erkennen. Zu diesem Zweck wird jedes Wort eines Namens mit der entsprechenden Wortart markiert; das verwendete Werkzeug sei im Folgenden als „Wortart-Markierer“ benannt, auch bekannt als Part-Of-Speech-Tagger (POS-Tagger) [Man03]. Um diese Aufgabe zu lösen, gibt es diverse Ansätze: Markov-Modelle, Hidden-Markov-Modelle, probabilistische, quantitative und viele weitere. Was all diese Methoden gemein haben, ist die Tatsache, dass für die Markierung (Tagging) immer nur das jeweilige Wort und dessen Wortumgebung (links, rechts oder bidirektional) betrachtet wird. Dies steht im Gegensatz zu lexikalischen Analysatoren, die komplette Sätze und deren Struktur bestimmen. Der Vorteil eines einfachen Wortart-Markierers ist die stabilere und schnellere Berechnung der Wortarten und die bessere Eignung für Wortgruppen und Einzelwörter. Des Weiteren ist die Erkennung der Wortart für die hiesige Aufgabenstellung ausreichend.

In dieser Arbeit wird der Part-Of-Speech-Tagger der Stanford University in der Version 3.1.3 Verwendung finden [TM00][TKMS03]. Dieser nutzt zur Erkennung der Wortart die Maximum-Entropie-Methode, das heißt, unter mehreren möglichen Modellen wird das mit der höchsten Entropie gewählt. In der neusten Version bietet der Stanford-POS-Tagger neben einem unidirektionalen auch einen bidirektionalen Markierungsalgorithmus, wofür unterschiedliche Modelle zur Verfügung gestellt werden. Für die vorliegende Arbeit wurde das Modell „english-bidirectional-distsim“ gewählt. Wie der Name schon sagt, arbeitet dieses Modell nur korrekt für englische Phrasen. Dies hat allerdings den Vorteil, dass sprachübergreifende Mehrdeutigkeiten ausgeschlossen werden können. Ein mehrsprachiges Modell hätte für das Beispielwort „hat“ keine Möglichkeit, festzustellen, ob es sich um die deutsche konjugierte Form des Verbs „haben“ handelt oder aber um das englische Substantiv „hat“ (Hut). Folglich ist im rein englischsprachlichen Kontext ein Model für englische Wörter zu bevorzugen. Die zweite Besonderheit dieses Modells ist die Bidirektionalität. Bei Tests hat sich herausgestellt, dass gerade für kurze Phrasen eine beidseitige Betrachtung deutlich bessere Ergebnisse liefert. Ähnlich gute Markierungen werden lediglich mit dem Modell „english-left3words-distsim“ erzielt; dieses betrachtet für jedes Wort die drei vorangegangen. Wortgruppen bestehen jedoch zumeist aus weniger als vier Einzelwörtern, was dieses Modell für die hiesige Arbeit unbrauchbar macht. Der Nachteil des genutzten bidirektionalen Modells ist, dass bei einigen Versuchsumgebungen der Java-Heap-Space vergrößert werden musste. Eine weitere Besonderheit des Stanford-POS-Taggers ist folgende: Unbekannte Wörter werden grundsätzlich als Substantiv markiert. Weiterhin erkennt der Stanford-POS-Tagger konjugierte Verben, Zeitformen, Fremdwörter und sonstige Sonderfälle, was wiederum zu einer hohen Genauigkeit führt. Den Beispielsatz „This is just a little test!“ markiert der Stanford-POS-Tagger wie folgt: „This_DT is_VBZ just_RB a_DT little_JJ test!_NN“. Die Markierungen seien in Tabelle 2.1 erläutert. Eine vollständige Auflistung befindet sich im Anhang A.

Tabelle 2.1.: Bedeutung der Markierungen des Stanford POS-Taggers

Markierung	Bedeutung
DT	Determinativ[2]
VBZ	Verb, dritte Person Singular, Präsens
RB:	Adverb
JJ:	Adjektiv
NN:	Substantiv, Singular

Um die eigentliche Aufgabe der Auffindung von Synonymen und Umschreibungen für die zuvor erwähnten sprechenden Namen der *Alice*-Objekte lösen zu können, wird eine Art lexikalische Datenbank benötigt, die zu einem gegebenen Wort ein Liste von Synonymen bestimmt. Diese sollten innerhalb eines semantischen Konzeptes zusammengefasst sein, das heißt, sollten für ein Wort mehrere Bedeutungen existieren, werden die Synonyme um die jeweiligen Bedeutungen gruppiert. Mit dieser Forderung geht die Unterscheidung der Wortart, die den jeweiligen „Synonym-Konzepten“ zugeordnet ist, einher. Ein hierarchischer Aufbau der Datenbank wird benötigt, um später eine geeignete Auswahl der tatsächlich im Kontext zutreffenden Synonyme zu erhalten.

WordNet ist eine solche Datenbank, speziell konzipiert als Wörterbuch im Programmierumfeld. Das hiesige Programm in seiner aktuellen Implementierung nutzt WordNet in der Version 2.1, da die Version 3 bisher nur Beta-Status erreicht hat. Die genutzte WordNet-API ist zwar grundsätzlich kompatibel, Zugriffsprobleme können jedoch nicht ausgeschlossen werden. Eine spätere Integration von WordNet 3.0 sollte aber problemlos möglich

[2] Determinative: bestimmte und unbestimmte Artikel, sowie adjektivische Determinativpronomina (dieser, mein, einige usw.)

sein. WordNet bietet folgende Funktionalität: Englische Substantive, Verben, Adjektive und Adverbien werden in sogenannten Synsets organisiert. Diese Synsets bestehen aus Wörtern, die unter einem bestimmten Konzept zusammengefasst werden können, folglich eine gleiche oder ähnliche Bedeutung haben. Ein bestimmtes Wort kann dabei in mehreren Synsets vorkommen. So bildet sich ein Netz von Synonymen, welches zudem stark hierarchisch ausgerichtet ist. Gleichzeitig bietet WordNet viele zusätzliche Funktionen: Es kann ebenso nach Antonymen, also Gegenteilen, wie „nass“ und „trocken“, und Hyponymie, also Unterordnung (auch „IS-A“-Relation) gesucht werden. Dies begründet auch die hierarchische Struktur von WordNet. Stellt man sich WordNet als Baumstruktur vor, so wird durch Hyponomie die Baumtiefe bestimmt, wohingegen die Synonyme die Baumbreite bestimmen. Ein Beispiel für Hyponomie ist die Hierarchiekette „Pflanze - Baum - Fichte - ...“.

Die Suche nach morphologisch variierten Wörtern, also Wörtern die durch Flexion, Konjugation, Anfügen von Vorsilben oder ähnlichem aus einer Basiswortform entstanden sind, ist ebenso möglich. Genauso können auch Wörter derselben Wortfamilie in einer anderen Wortart bestimmt werden, also beispielsweise das Verb „belegen“ für das Ursprungswort „Beleg“ (Substantiv).

WordNet verfügt noch über weitere interessante Funktionen, die allerdings für den hiesigen Kontext keine Relevanz haben und deshalb nicht weiter betrachtet werden sollen. Um einen Überblick über den vollen Funktionsumfang von WordNet zu erhalten, bieten sich die Artikel von Miller [Mil95] und Pedersen [PPM04] an. Die interne Struktur von WordNet sei jedoch noch erläutert: Jedes Synset hat in WordNet eine eindeutige Kennung, die aus einer Zahl gefolgt von den Wörtern, durch die das Synset bestimmt wird, besteht. Die Kennung allein genügt zur Identifikation des Synsets innerhalb von WordNet. Allerdings ändern sich die internen Kennungen bei jeder neuen WordNet-Version, weshalb an dieser Stelle explizit auf die Version 2.1 verwiesen wird. Für die Funktionalität des im Zuge der Studienarbeit erstellten Programms hat diese Varianz keine Auswirkung. Allerdings werden die Kennungen im letzten Programmschritt (siehe Kapitel 4.10 und 5.8) in der Ontologie gespeichert. Dies hat zur Folge, dass eine Ontologie, die zuvor mit WordNet 2.1 um Synonyme erweitert wurde, keine korrekten Kennungen enthält, sollte auf die selbe Ontologie später mittels einer anderen WordNet-Version zugegriffen werden. Folglich empfiehlt es sich, zur Anfrage ebenfalls WordNet 2.1 zu nutzen oder gegebenenfalls die Synonymerweiterung unter Einbeziehung der gewünschten Version erneut durchzuführen.

Die Abbildung 2.3 zeigt einen Ausschnitt aus WordNet zum Begriff „car“ mit drei unterschiedlichen Bedeutungen. Die aufwärts gerichteten Pfeile symbolisieren die Hyponym-Beziehung zwischen den Synsets. Die als Kästen dargestellten Synsets beinhalten die Wortart, die WordNet-Kennung, sowie die Wörter die dem gleichen Konzept zugeordnet werden.

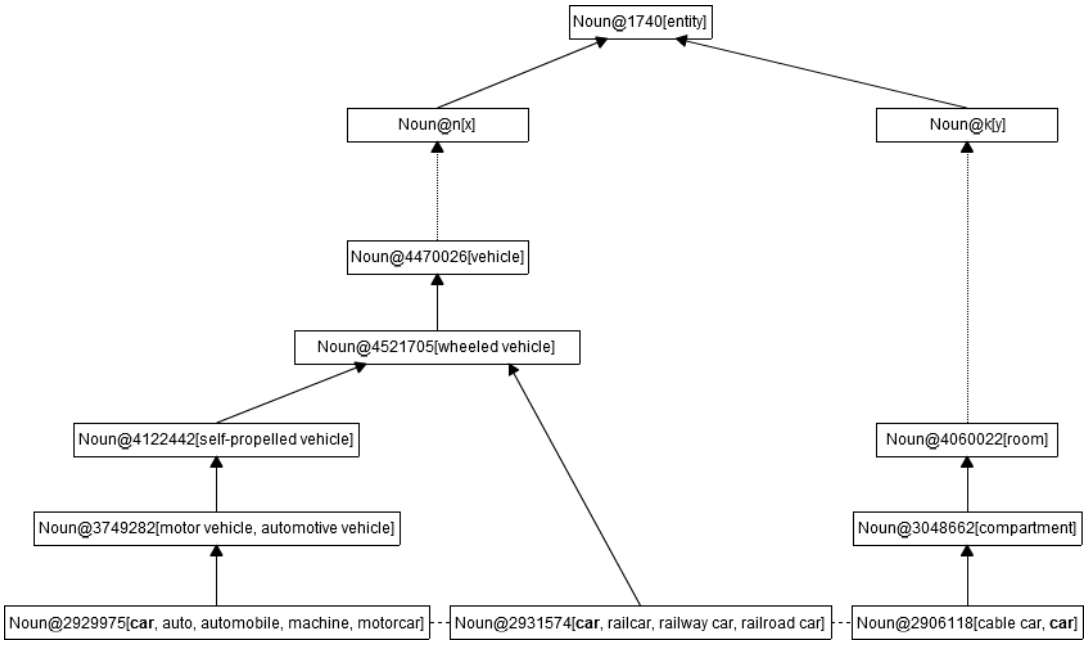

Abbildung 2.3.: Hierarchische Struktur von WordNet

3. Verwandte Arbeiten

An dieser Stelle sollen einige Arbeiten, die den Fachbereich der natürlichsprachlichen Programmierung charakterisieren, vorgestellt werden. Diese sollen stellvertretend für die Geschichte und den aktuellen Stand der Forschung in diesem Gebiet stehen. Gleichwohl wurden diese Projekte ausgewählt, um einen Überblick über die unterschiedlichen Herangehensweisen zu zeigen und die grundlegende Andersartigkeit des hier vorgestellten Projekt herauszustellen. Anschließend werden Arbeiten gezeigt, die eine ähnliche Zielsetzung wie die hier dargelegte Studienarbeit haben. Diese befassen sich entweder mit der lexikalischen Analyse und Erweiterung von Ontologien oder mit Synonymfindung im Allgemeinen.

3.1. Natürlichsprachliche Programmierung

Metafor [LL05] erzeugt aus einem Text, der in Form einer Geschichte geschrieben ist, Quelltext. Somit ist die zugrundeliegende Idee der in diesem Projekt verfolgten sehr ähnlich. Metafor analysiert den Text und erzeugt daraus Objekte, Methoden, Variablen und weitere typische Programmierkonzepte. Diese bleiben allerdings, im Gegensatz zur Zielstellung unseres Projektes, ohne Inhalt. Somit bietet Metafor ein leistungsfähige Möglichkeit, die Semantik eines Programms zu beschreiben und Gedanken und Ideen in eine schematische Form zu übertragen, die sich gleichzeitig an die Syntax der jeweiligen Programmiersprache hält. Folglich eignet sich Metafor hauptsächlich als Entwurfswerkzeug.

Das **Pegasus**-Projekt der Universität Darmstadt [KM06] verfolgt ebenfalls das Ziel, natürliche Sprache in Programmcode umzusetzen. Der Ansatz ist indes ein völlig anderer. Der analysierte Text wird in eine sogenannte „Idea Notation“ überführt. Ein Beispiel ist innerhalb der Arbeit für die Begriffe „wood“ und „table“ gegeben:

```
wood   :  [smell of wood, warmth, solidness brown color]
table  :  [horizontal panel, vertical bars, wood, solidness]
```

Diese Konzepte bilden folglich ein Netz von Wörtern, ähnlich dem durch Synsets erzeugten Netzes in WordNet. Allerdings ist hier die Verknüpfung unter dem abstrakten Begriff „Idea“ (Idee) geformt. Ein aus *Ideen* und Verknüpfungen entstandenes Netz ist in Abbildung 3.1 zu sehen. Diese semantische Darstellung und die erzeugten Verknüpfungen

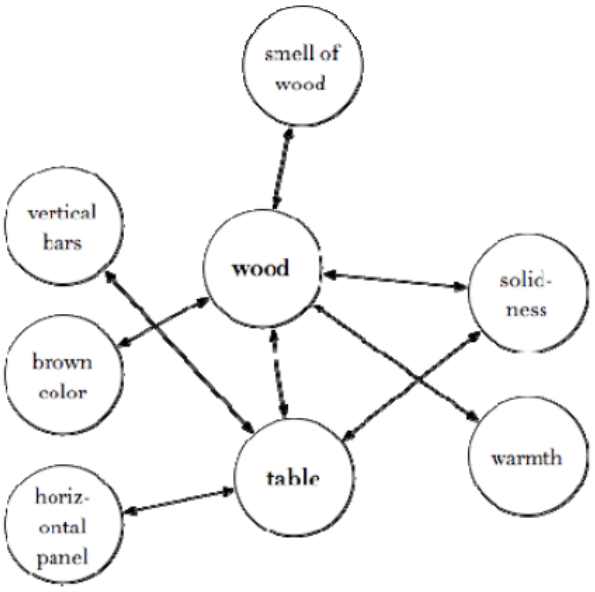

Abbildung 3.1.: Idea Notation aus dem Pegasus-Projekt

bilden das Kernstück von Pegasus. Die *Ideen* werden schließlich auf Programmcode abgebildet. Der Vorteil der „Idea Notation“ ist ganz eindeutig die Universalität. Theoretisch können *Ideen* auf jede beliebige Programmiersprache abgebildet werden. Gleichzeitig unterstützt Pegasus bereits sowohl deutsche als auch englische Texte. Ein weiterer Vorteil der „Idea Notation“ ist, dass Texte, die in unterschiedlichen Sprachen formuliert wurden, auf die gleiche *Idee* abgebildet werden können, wenn geeignetes Hintergrundwissen (wie z.B. den eigens angelegten Wörterbüchern, die für jedes Wort die entsprechende Aktions/Verb-, Entitäts/Substantiv- und Eigenschafts/Adjektiv-Form bereithalten) Verwendung findet. Allerdings ist auch Pegasus in seiner Mächtigkeit noch sehr beschränkt. Die Sprachanalyse funktioniert zwar, wie auch der Aufbau der „Idea Notation“, sehr gut, jedoch kann diese bisher nur auf ein Teil der üblichen Programmierkonzepte abgebildet werden. Komplexe Programmstrukturen sind somit noch nicht natürlichsprachlich beschreibbar. Gleichzeitig zeigt Pegasus einen interessanten Ansatz zur natürlichsprachlichen Programmierung auf, bei gleichzeitig beachtlichem Funktionsumfang.

Natural Java [PRZH00] ist ein Prototyp eines intelligenten Java-Interfaces, welches mittels natürlicher Sprache kontrolliert wird. Mit Natural Java ist es sowohl möglich, neuen Quelltext zu erstellen als auch vorhanden zu bearbeiten. Der Nutzer bekommt dabei dauerhaft Rückmeldung über die Umsetzung seiner Eingaben. Dies stellt einen großen Vorteil gegenüber vergleichbaren Ansätzen dar, da Fehler direkt entdeckt werden und der Programmierer jederzeit Überblick über den aktuellen Fortschritt der Arbeit erhält. Andererseits ist die Spracherkennung nicht besonders leistungsstark, was die Ausdrucksmöglichkeiten sehr stark einschränkt und somit den natürlichsprachlichen Ansatz sehr programmiersprachenähnlich erscheinen lässt.

NLC [BB79] war ein sehr früher Versuch, sich dem Problem der natürlichsprachlichen Programmierung zu nähern. Bereits 1979 beschreiben Bruce W. Ballard und Alan W. Biermann diesen, von ihnen entwickelten, domänenspezifischen Prototypen. NLC ist zunächst nur für Matrizen- und Tabellenkalkulationen geeignet, kann aber den Autoren zufolge auf andere Domänen erweitert werden. Neben dieser Einschränkung gibt es Vorgaben für die Form der Nutzerbefehle; so müssen alle Sätze (Befehle an das Programm) im Imperativ mit dem Verb zu Beginn des Satzes formuliert werden. Diese Einschränkungen

vereinfachen die Problematik enorm. Folglich funktioniert der Prototyp beinahe fehlerfrei, obwohl als Hintergrundwissen nur ein mit 450 Einträgen recht kleines Wörterbuch zur Verfügung steht. Gleichzeitig erhält der Nutzer ähnlich wie bei Natural Java [PRZH00] ein direktes Feedback. Trotz all seiner Beschränktheit ist NLC ein erstaunlicher Erfolg unter Beachtung der Entstehungszeit und besticht durch drei wichtige Einsichten im Bezug auf natürlichsprachliche Programmierung:

- eine Nachfrage an natürlichsprachlichen Programmierlösungen ist eindeutig gegeben,
- eine Art Rückmeldung/Feedback an den Programmierer ist sowohl unabdingbar als auch erwünscht,
- um eine leistungsfähige Lösung etablieren zu können, ist es nötig, gewisse (domänenspezifische) Einschränkungen zu fordern, auch wenn diese wenn möglich nicht so deutlich ausfallen sollten wie bei NLC.

Alle hier vorgestellten Arbeiten abstrahieren seitens des natürlichsprachlichen Textes ein Konzept oder eine Idee; das heißt, aus dem Text wird auf die eine oder andere Weise ein Modell erstellt (Schritt 1 in der Abbildung 3.2). Dieses wird dann in Quelltext der jeweiligen Programmiersprache umgesetzt (Schritt 2). Das hiesige Projekt, das vergleichend in Abbildung 3.3 dargestellt ist, verfährt anders: die Programmiersprache, hier *Alice*, wird modelliert (Schritt 1 in der Abbildung 3.3). Anschließend wid zunächst versucht, den Text auf das Modell abzubilden (Schritt 2). Erst dann wird, ausgehend von dem Modell und den verknüpften Objekten und Methoden, der *Alice*-Quelltext/Skript erstellt (Schritte 3 und 4). Durch dieses Vorgehen bleibt die Arbeit innerhalb eines vorgegeben Rahmens, wodurch das Gesamtprojekt beherrschbar bleiben und vorhersehbare Ergebnisse liefern soll.

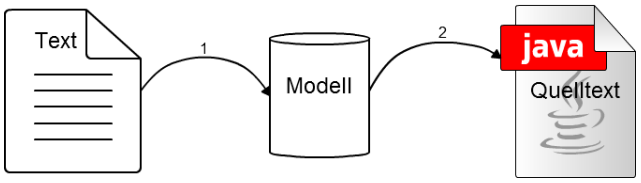

Abbildung 3.2.: Herkömmlicher Ansatz

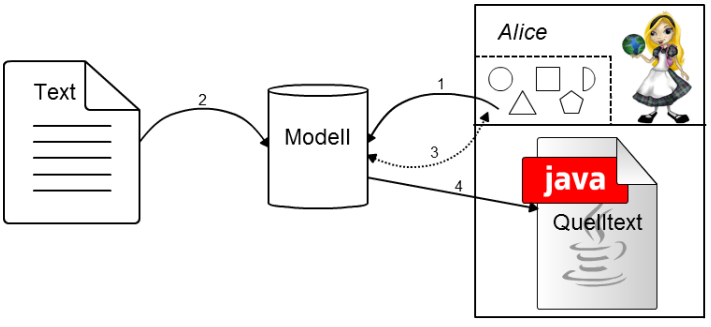

Abbildung 3.3.: Ansatz für das Projekt „Programmieren in natürlicher Sprache“

3.2. Lexikographische Untersuchung und Erweiterung von Ontologien

Arbeiten mit derselben Zielsetzung, wie die hier vorgestellte, konnten im Zuge der Literaturrecherche nicht identifiziert werden. Erweitern von Ontologieelemente um Synonyme

war bisher kaum Gegenstand der Forschung. Es beschäftigen sich jedoch einige Arbeiten mit der Auffindung von Synonymen im Ontologieumfeld. Andere nutzen Synonyme zur Identifizierung beziehungsweise Gleichsetzung von Ontologieelementen. Weitere nutzen Synonyme zur Erweiterung anderer Daten, wie zum Beispiel Anfragen an Datenbanken oder Ähnliches.

Ein solches Anfrage-System wird von Liu et. al in der Arbeit **An Effective Approach to Document Retrival via Utilizing WordNet and Recognizing Phrases** beschrieben [LLYM04]. Über ein mehrstufiges Verfahren wird versucht, Suchanfragen zum Auffinden von Dokumenten lexikographisch zu erweitern, um eine höhere Auffindungsquote zu erzielen. Unter anderem sollen hierfür die Anfragen um Synonyme beziehungsweise Hyponyme erweitert werden. Wie auch in der hiesigen Arbeit, wird entschieden welche Wörter für eine Synonymsuche relevant sind. Zunächst wird ein Part-Of-Speech-Tagger verwendet, um die Wortart der Einzelwörter zu bestimmen. Die Synonymsuche wird dann auf Substantive beschränkt. Unter den möglichen Substantiven wird dann erneut mithilfe unterschiedlicher Techniken gewählt. So wird unter anderem ein Wörterbuch zu Rate gezogen, um bekannte Phrasen beziehungsweise zusammengesetzte Substantive zu detektieren. Weiterhin wird die Länge und Art der Anfrage, sowie die Position der Substantive innerhalb der Anfrage betrachtet. Wurde eine relevante Substantiv-Menge bestimmt, so werden für diese Substantive sowohl Synonyme, als auch Hyponyme (siehe Kapitel 2) und deren Definitionen mittels WordNet bestimmt. Unter dieser Synonym-Menge wird dann wiederum ausgewählt. Hierzu wird ein mehrstufiges Verfahren verwendet, welches unter anderem die Auftretenswahrscheinlichkeit als auch das gleichzeitige Auftreten von Wörtern in mehreren Bedeutungen (Synsets) oder Hyponym-Mengen betrachtet. Das grundlegende Vorgehen ist somit vergleichbar mit dem hier gewählten Ansatz. Zunächst werden die Wortarten der Einzelworte der Phrase bestimmt, dann die relevanten Worte ausgewählt, bevor Synonyme bestimmt werden. Zuletzt wird unter den Synonymen ausgewählt. Es zeigen sich jedoch auch deutliche Unterschiede: Zum einen wird in Kapitel 4.5 gezeigt werden, dass eine Beschränkung auf Substantive für das relevante Wort im hiesigen Kontext nicht sinnvoll ist. Weiterhin wird auch die Synonym-Auswahl, wie in Kapitel 4.9 beschrieben, anders vorgenommen: Statt Wahrscheinlichkeiten und Wortüberschneidungen wird die hierarchische Struktur von WordNet und der zugrundeliegenden Ontologie, die die *Alice*-Welt modelliert, genutzt. Die Ontologie stellt zusätzliches Wissen zur Verfügung, die Liu et. al bei der Verarbeitung einer Suchanfrage nicht zur Verfügung stehen. Trotzdem konnte in dieser Arbeit gezeigt werden, dass durch Hinzufügung von Synonymen zu einer Anfrage, abhängig vom zu suchenden Dokument, eine Steigerung der Auffindungsquote von 4% bis 34% erzielt wird. Auch wenn die Arbeit von Liu et. al nicht gänzlich mit der hiesigen vergleichbar ist, da Suchanfragen statt einer Ontologie betrachtet werden, lassen die erzielten Ergebnisse darauf schließen, dass Synonym-Erweiterung zur Auffindung von Daten sinnvoll ist.

Eine Arbeit, die Synonyme im Ontologie-Umfeld nutzt, ist **Semi-Automatic Ontology Extension Using Spreading Activation** von Wei Liu et. al [LWSC08]. Allerdings werden Ontologien in diesem Fall nicht um Synonyme erweitert. Vielmehr wird Synset-Bildungs mittels WordNet genutzt um für mögliche Erweiterungen zu prüfen, ob diese nicht redundant oder bereits in der Ontologie vorhanden sind. Dies geschieht wie folgt: Domänenspezifische Daten werden aus dem Internet gesammelt und anschließend lexikalisch untersucht, das heißt, es wird überprüft, ob es Überschneidungen beziehungsweise Redundanzen innerhalb der gesammelten Daten gibt. Die so vorverarbeiteten Daten werden als mögliche Erweiterung angesehen. Unter Einbeziehung der Wortart – wieder werden lediglich Substantive betrachtet – werden Vergleiche mit den Daten innerhalb der vorhandenen Ontologie angestellt und somit potentielle Erweiterungen bestimmt. Erweiterungen können hierbei sowohl neue als auch Verfeinerungen bekannter Konzepte der betrachteten

Ontologie sein. Schließlich wird WordNet genutzt um zu bestimmen, ob die neuen Daten tatsächlich neue Informationen enthalten oder lediglich Synonyme bereits existierender Ontologie-Elemente sind. Dieser Ansatz ist jedoch nur semi-automatisch, da die gesammelten Erweiterungskandidaten noch von Domänenspezialisten überprüft werden müssen bevor sie tatsächlich in die Ontologie eingefügt werden. WordNet wird hier also im Gegensatz zur hiesigen Arbeit zum Ausschluss genutzt. Hierfür werden, ähnlich wie später in Kapitel 4.9 erläutert, die hierarchischen Strukturen von Ontologien und WordNet genutzt. Es wird also versucht über Hierarchien zusätzliche Informationen zu gewinnen, um Synonyme besser zu erkennen und auszuschließen.

Einen ähnlichen Ansatz zu einem anderen Thema liefern Jayant Krishnamurthy und Tom M. Mitchell in ihrer Abhandlung **Which Noun Phrase Denote Which Concepts?**. Sie beschäftigen sich allgemein mit der Bestimmung von Substantiv-Synonymen ohne Einschränkung des Kontextes. Das besondere Augenmerk liegt hier auf der Auflösung von Mehrdeutigkeiten. Hierzu wird versucht, durch Nutzung von Hintergrundwissen Konzepte zu einem Wort zu erstellen, die tatsächlich in den Kontext passen. Die Konzepte enthalten bedeutungsgleiche Wörter, Abkürzungen, abgeleitete Wortformen und, falls passend, Hyperonyme (siehe Kapitel 5.7). Diese Informationsvielfalt unterscheidet die in dieser Arbeit eingeführten Konzepte von den Synsets in WordNet, die ebenfalls eine Art Konzept darstellen. Letztere fassen lediglich Wörter mit der gleichen Bedeutung zusammen. Wichtiger als dieser Unterschied ist jedoch das Vorgehen, um die Informationen zu erhalten: Für alle möglichen Bedeutungen eines Wortes, falls Mehrdeutigkeiten existieren, wird mithilfe von Hintergrundwissen überprüft, ob diese Bedeutung im Kontext sinnvoll ist. Das Hintergrundwissen bildet eine Ontologie und einen Satz zuvor markierter Synonyme. Kann im Kontext des betrachteten Wortes ein Wort gefunden werden, das in der Ontologie mit dem betrachteten Wort verknüpft ist, so wird diese Bedeutung als die richtige angesehen. Als Beispiel dient in der Arbeit das Wort „apple“ (Apfel) mit den möglichen Bedeutungen „apple - the fruit“ (Apfel - die Frucht) und „Apple Computer“ (Rechner der Firma Apple). Im Kontext von „apple“ kann im Beispiel die Phrase „Steve Jobs“ erkannt werden, weshalb die zweite Bedeutung als die richtige angenommen wird.

Auch wenn keine vergleichbare Arbeit existiert, so sind einige Ansätze ähnlich. Besonders die Auswahl der Synonyme über Hintergrundwissen, das zumeist über eine Ontologie zur Verfügung gestellt wird, wird aufgegriffen, jedoch anders realisiert (siehe hierzu Kapitel 5.7).

4. Analyse und Entwurf

Der Entwurf der Arbeit unterliegt drei analytischen Grundfragen bezüglich der Rolle der Arbeit im Kontext des Gesamtprojekts:

(1) Welche (Teil-)Aufgabe soll im Rahmen des Gesamtprojekts gelöst werden?

(2) Welchen Nutzen hat diese (Teil-)Lösung für das Gesamtprojekt?

(3) Welcher Ansatzpunkt wird zur Lösung der Aufgabenstellung gewählt?

Die Aufgabe (1), der sich die vorliegende Arbeit widmet, ist das Auffinden von Synonymen zu Individuen einer Ontologie[1]. Hierbei stellt sich eine besondere Herausforderung: Die Namen der Individuen sind, bedingt durch ihre Bedeutung für das Gesamtprojekt, im Allgemeinen Wortgruppen, also keine Einzelwörter. Zwar ist es möglich, den Synonymbegriff derart zu erweitern, dass er (Sinn-)Gleichheit zwischen Wortgruppen statt Einzelwörtern beschreibt, doch ist diese Aufgabe höchst komplex und eine technische Umsetzung schlichtweg (noch) nicht vorhanden. Deshalb wird an dieser Stelle die Annahme getroffen, dass jede Wortgruppe ein relevantes beziehungsweise sinngebendes Wort besitzt. In der Linguistik wird diese Annahme mit dem Terminus „head of phrase“ (Kopf der Phrase) beschrieben und unter anderem von Chomsky diskutiert [Cho95].

Im Gesamtkontext (2) sind die Individuen die Namen von *Alice*-API-Methoden oder Objekten. Mithilfe der Ontologie soll später ein *Alice*-Skript erstellt werden. Dies setzt aber voraus, dass ein gegebener Text korrekt mit der Ontologie verknüpft werden kann. Das Hinzufügen der Synonyme zu den Methoden- beziehungsweise Objekt-Namen soll Variationen in den natürlichsprachlichen Texten teilweise kompensieren. Im Folgenden seien Objekte im Sinne der Konformität mit dem Rahmenwerk als „Modelle“ bezeichnet.

Der Ansatz (3) über die Methoden- beziehungsweise Modell-Namen von *Alice* wird aus folgendem Grund gewählt: Es wird angenommen, dass diese Namen sogenannte „sprechende“ Methoden-Namen sind. Das heißt, es kann davon ausgegangen werden, dass der Methoden-Name bereits beschreibt was innerhalb der Methode geschieht. Gleiches gilt natürlich analog für Modell-Namen. Dieses Vorgehen ist eine softwaretechnische Konvention, welche innerhalb der *Alice*-API nahezu durchgängig Anwendung findet. Ein Beispiel aus *Alice* ist die Methode `ScratchEar`, die in diesem Fall einen Husky dazu bringt sich am Ohr zu kratzen.

[1] Zu beachten ist, dass nach lexikalischer (Sinn-)Gleichheit gesucht wird (Beispiel: „Flur“ - „Diele“ - „Korridor“), nicht nach Gleichheit von Individuen im ontologischen Sinn (Beispiel: „Theodor Heuss“ - „ehemaliger Bundespräsident“).

Im Zuge der Anforderungsanalyse wird eine exemplarisch erstellte *Alice*-Welt und die zugehörige Ontologie betrachtet. Beides stammt aus der Bachelorarbeit von Oleg Peters [Pet12]. Dort wurde versucht, die *Alice*-API in geeigneter Weise durch eine Ontologie zu modellieren. Dafür wurde eine *Alice*-Welt erstellt und mit einigen Objekten (Modellen) besiedelt, die bestimmte Aktionen (Methoden) durchführen. Daraus entsteht ein *Alice*-Skript, dass einen Ausschnitt der *Alice*-API beinhaltet. In der daraus erstellten Ontologie bilden die Namen der Modelle, Methoden und weiteren Objekte beziehungsweise Funktionen die Menge der Individuen. Diese wiederum bevölkern die Klassen der Ontologie; die Klassen beschreiben hierbei die Funktionalität des Individuums. Ein Individuum, das einen Modell-Namen trägt, wäre folglich in der Ontologie-Klasse „Modell“ angesiedelt. Dies gilt analog für Methoden und alle anderen Funktions- und Objekt-Typen. Der hierarchische Aufbau der betrachteten Ontologie wird im Kapitel 4.4 ausführlicher erläutert. Weiterhin bestehen Relationen zwischen Modellen und Methoden, zwischen Modellen und Posen, zwischen Modellen und Modellen und zwischen anderen Individuen der Ontologie derart, dass Beziehungen wie „besteht aus“ und „hat Methode“ modelliert wurden. So könnte zwischen dem imaginären Modell „Hund“ und dem ebenfalls imaginären Modell „Kopf“ eine „besteht aus“-Relation bestehen. Ein weiteres Beispiel wäre das Modell „Hase“, die Methode „springen“ und die Relation „hat Methode“. Die Relationen der Ontologie und deren Relevanz für den hiesigen Kontext wird in den Kapiteln 4.5 und 4.9 erläutert.

Die vorliegende *Alice*-Welt soll gleichzeitig als Grundlage für beispielhafte Beschreibungen einiger Entwurfsentscheidungen dienen. Unter dieser Zielsetzung kann nun ein entsprechendes Programm entworfen werden. Die Aufgabenstellung erfordert hierbei mehrere aufeinanderfolgende Arbeitsschritte, die im Aktivitäts-Diagramm (Abbildung 4.1) skizziert werden. Die Änderung am Namen des Beispiel-Individuums „LinkerObererArm“ kann dabei in den Annotationen der jeweiligen Programmschritte nachvollzogen werden. Wie in der Abbildung zu erkennen ist, wird zunächst der Name aus dem Individuum extrahiert und in ein für natürliche Sprache übliches Format überführt. Im nächsten Schritt werden mithilfe des Wortart-Markierers die Wortart eines jeden Wortes des Namens bestimmt und im Folgenden in einem eigens angelegten Objekt gespeichert. In diesem Objekt ebenfalls befindlich sind die Resultate der zwei folgenden Verarbeitungsschritte: Zunächst wird die Klassenzugehörigkeit (siehe Kapitel 2) des Individuums bestimmt. Mithilfe dieser zusätzlichen Information wird dann das für die Wortgruppe relevante Wort bestimmt. Im nächsten Schritt kann, falls nötig, die Wortart des relevanten Wortes angepasst werden. Mit den gesammelten Informationen (Wortart, Klassenzugehörigkeit) werden anschließend die Synonyme bestimmt. Sollten keine Synonyme in der Datenbank zu finden sein, lässt sich dies eventuell durch eine sequentielle Teilwortfindung beheben. Im vorletzten Schritt wird versucht, die nicht treffenden Synonyme auszusortieren, bevor im letzten Schritt die gefundenen Informationen zur existierenden Ontologie hinzugefügt werden. Zu beachten ist, dass alle Schritte für jedes Individuum der Ontologie wiederholt werden.

4.1. Namensextraktion

Zunächst muss, wie beschrieben, jedes Individuum beziehungsweise dessen Name einzeln aus der Ontologie extrahiert werden. Die Namen sind dabei wie zuvor erläutert „sprechend“. Allerdings sind die Wortgruppen noch nicht als solche erkennbar, da diese, wie im Programmierumfeld üblich, in Binnenmajuskel-Schreibweise gehalten sind. Binnenmajuskel sind Großbuchstaben die im Wortinneren vorkommen. Im Rahmen der Softwaretechnik wird diese Syntaktik zur Separierung von Einzelwörtern innerhalb von Namen genutzt. Beispiele aus der betrachteten *Alice*-Welt finden sich mit der Methode `SitStandWalk` in Tabelle 4.1 oder dem Modell `LeftUpperArm` im Aktivitätsdiagramm (Abbildung 4.1). Diese Schreibweise muss zunächst in die für natürliche Sprache übliche, mit Leerzeichen zwischen den Einzelwörtern, überführt werden. Dies ist nötig, da Wortart-Markierer im

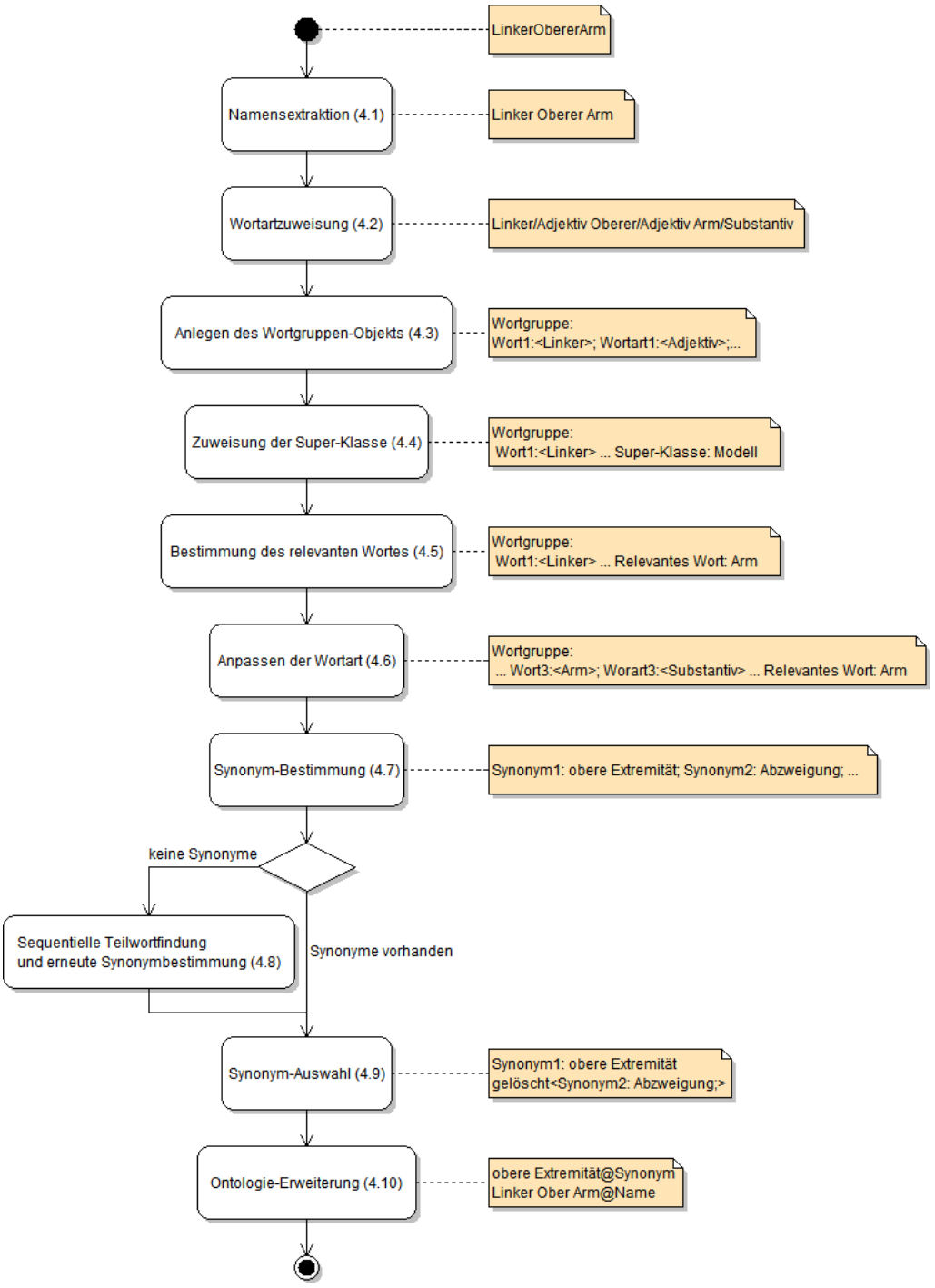

Abbildung 4.1.: Entwurf des Programmablaufs

Allgemeinen nur Sätze und Wortgruppen in natürlichsprachlicher Schreibweise verarbeiten können. Dies gilt auch für den hier eingesetzten Stanford Part-Of-Speech-Tagger. Bei der Umsetzung von Binnenmajuskel zu natürlicher Schreibweise gilt es diverse Ausnahmen zu beachten, die in der nicht immer konsequenten Umsetzung der Binnenmajuskel-Konvention ihren Ursprung haben. Die Tabelle 4.1 zeigt alle möglichen Schreibweisen der extrahierten Namen und deren natürlichsprachlichen Entsprechungen. Wie zu erkennen ist, gibt es Fälle, bei denen ein Unter- oder Bindestrich zur Separierung genutzt wird, oder an Objekt-Namen wird eine Zahl angehängt, sollte es mehrere Objekte des gleichen Typs innerhalb der *Alice*-Welt geben. Diese Ausnahmen sind einfach zu behandeln. Es gibt jedoch auch

Fälle, bei denen gänzlich auf eine Separierung, sei es durch Binnenmajuskel-Schreibweise oder Unter- beziehungsweise Bindestriche, verzichtet wurde, wie in der letzten Zeile der Tabelle durch das Individuum `uglyfish` beispielhaft belegt. Hier kann zunächst keine Aufteilung in Einzelwörter vorgenommen werden. Derartige Individuen-Namen stellen ein enormes Problem bei der späteren Suche nach Synonymen dar. Trotzdem soll diese Art Ausnahme, nach intensiver Problemanalyse, im Programmablauf erst behandelt werden, wenn für die Wortgruppe auch nach Durchführung andere Ausnahmebehandlungen kein Synonym gefunden werden konnte. Eine ausführliche Erläuterung dieser Problematik befindet sich in Kapitel 4.8.

Tabelle 4.1.: Namensextraktion und Umwandlung in das natürlichsprachliche Format

Extrahierter Name	Natürlichsprachliches Format
Husky	Husky
SitStandWalk	Sit Stand Walk
body-front	body front
drive_right	drive right
Penguin2	Penguin 2
uglyfish	uglyfish

4.2. Wortartzuweisung

Die im natürlichen Sprachformat vorliegenden Wortgruppen können nun hinsichtlich der Wortarten der Einzelwörter untersucht werden; diese Aufgabe übernimmt der Wortart-Markierer. Die entsprechende Wortart wird dem jeweiligen Wort in geeigneter Weise angehängt. Ein anschauliches Beispiel für die Markierung durch den Stanford Part-Of-Speech-Tagger wurde mit „This_DT is_VBZ just_RB a_DT little_JJ test!_NN“ bereits in Kapitel 2 gezeigt. Die Wortarterkennung ist unabdingbar, da später unter anderem auf dieser Grundlage das für die Wortgruppe relevante Wort bestimmt wird.

4.3. Wortgruppen-Objekt

Die Wortgruppe sollte nun in einem Format gespeichert werden, das für die weitere Verarbeitung geeignet ist. Es sollte möglich sein, einzeln auf die Wörter zuzugreifen, ohne die gesamte Wortgruppe zu betrachten. Weiterhin sollte die Verknüpfung von Wort und Wortart erhalten bleiben. Objektorientierte Programmiersprachen bieten diverse Konstrukte für derartige Problemstellungen an; so wäre eine Abbildung von Wort zu Wortart denkbar oder ein zweidimensionales Feld. Im vorliegenden Fall ist es jedoch von Vorteil, gemeinsam mit der Wortgruppe Zusatzinformationen zu speichern. Hierzu gehören unter Anderem die Anzahl der Wörter, die übergeordnete Klasse (siehe Kapitel 4.4), sowie das relevante Wort der Wortgruppe (siehe Kapitel 4.5). Aus diesem Grund bietet sich die Erstellung eines eigenen Objekts für die Wortgruppe an, das neben den Wörtern mit den zugehörigen Wortarten alle Zusatzinformationen speichert. Im Aktivitätsdiagramm (Abbildung 4.1) ist das Einfügen des ersten Wortes der Wortgruppe, sowie dessen Wortart, skizziert. Weitere Vorteile dieses Ansatzes sind der direkte und intuitive Zugriff auf alle Elemente[2] und die Möglichkeit, dem Objekt jederzeit weitere Attribute hinzuzufügen.

4.4. Super-Klasse

Innerhalb des Projekts „Programmieren in natürlicher Sprache“ sind die von Oleg Peters [Pet12] erstellten Ontologien, die die *Alice*-API modellieren, wie zu Beginn von Kapitel 4 erläutert, strukturiert: Individuen bevölkern Klassen beziehungsweise Super-Klassen,

[2] Dies ist sowohl bei Abbildungen als auch bei Feldern nicht ohne Weiteres möglich.

die die Funktionalität der beinhaltenden Individuen beschreiben. So ist zum Beispiel das Individuum `ScratchEar` in der Ontologie innerhalb der Klasse „UserDefinedMethod“ angesiedelt oder `LeftUpperArm` ein Individuum der Klasse „Model“. Ebenfalls zu beachten ist, dass jedes Individuum nur einer Klasse angehören kann. Die meisten Ontologie-Implementierungen würden zwar auch die Möglichkeit bieten, ein Individuum innerhalb mehrerer Klassen existieren zu lassen, doch wäre dies für die Modellierung der *Alice*-API nicht hilfreich. Offensichtlich kann eine Methode nicht gleichzeitig ein Modell sein, oder eine Individuum der Klasse „Pose“ außerdem der Klasse „Kamera“ angehören.

Das eigentliche Ziel ist aber, nicht nur die Klasse zu bestimmen, der das betrachtete Individuum angehört, sondern, falls möglich, mehrere Klassen einer Super-Klasse zuzuordnen und somit die Weiterverarbeitung zu generalisieren. Dies ist durch die hierarchische Anordnung von Klassen in Ontologien und der zuvor erläuterten eindeutigen Klassenzugehörigkeit der Individuen im vorliegenden Fall möglich. So gehören unter anderem in der betrachteten Ontologie die Klassen „Model“ und „FirstClassModel“ der Super-Klasse „AliceModel“ an. Gleichzeitig sind die Individuen der Subklassen im Sinne der Synonymauffindung kongruent. Somit kann ihnen die gleiche Super-Klasse zugeordnet werden[3]. Für das Auffinden der Super-Klasse bietet sich eine rekursive Funktion an. Die gefundene (Super-)Klassenzugehörigkeit wird im Wortgruppenobjekt hinterlegt. In Abbildung 4.2 ist die Klassenhierarchie der betrachteten Ontologie, visualisiert als Baum, dargestellt. Eine kurze Erläuterung zu den Bedeutungen der Klassen befindet sich in Anhang B.

4.5. Das relevante Wort

Mit den nun vorliegenden Informationen über Super-Klassen-Zugehörigkeit und Wortart der Einzelwörter kann das relevante oder sinngebende Wort der Wortgruppe gefunden werden. Der linguistische Terminus „head of phrase“ wurde bereits zu Beginn des Kapitels 4 eingeführt. Die Definition für das relevante Wort der Wortgruppe ist jedoch nicht in allen Fällen eindeutig. Aus diesem Grund finden sich in der Literatur viele unterschiedliche Formulierungen. Für den hiesigen Kontext ist vor allem folgende, von Miller dargelegte, relevant [Mil11]: „The basic criteria for recognizing the head of a syntactic construction are that the head is obligatory and controls certain other possible constituents (words) called modifiers“; das heißt, das wichtigste Kriterium, um den Kopf einer Phrase (das relvante Wort) zu bestimmen, ist zum einen, dass dieser „Kopf“ für die Wortgruppe zwingend erforderlich ist und zum anderen, dass dieser ein oder mehrere Wörter kontrolliert, die das relevante Wort modifizieren. Ein Wort innerhalb der Wortgruppe zu finden, welches diese Kriterien erfüllt, gestaltet sich aber aufgrund der wenigen vorliegenden Informationen und der Kürze der Wortgruppen als schwierig. Würden vollständige Sätze statt Wortgruppen betrachtet, so könnte zum Beispiel mithilfe eines lexikalischen Analysators die Satzstruktur bestimmt und als sinngebendes Wort möglicherweise das Subjekt angenommen werden. Dies ist bei Wortgruppen, die zumeist nur aus wenigen Wörtern bestehen, so natürlich nicht möglich. Modelle und Methoden sind die Klassen mit den meisten Individuen; innerhalb dieser Klassen treten für die hier betrachtete *Alice*-Welt folgende Wortkonstellationen am Häufigsten auf:

- für Modelle: <Adjektiv>+<Substantiv/Subjekt>
- für Methoden: <Verb>+<Substantiv/Objekt>

Die Wortgruppen bestehen also zumeist nur aus zwei Wörtern. Aus diesem Grund können lediglich die Funktionalität innerhalb von *Alice* (angegeben durch die Super-Klasse) und die Wortart der Einzelwörter zur Bestimmung des relevanten Wortes beitragen.

[3] Dies gilt nicht für die Klassen „AliceMethod“ und „UserDefinedMethod“, die zwar beide der Super-Klasse „Method“ angehören, also jeweils Methoden beschreiben, jedoch unterschiedlich behandelt werden müssen. Sie hierzu Kapitel 5.2

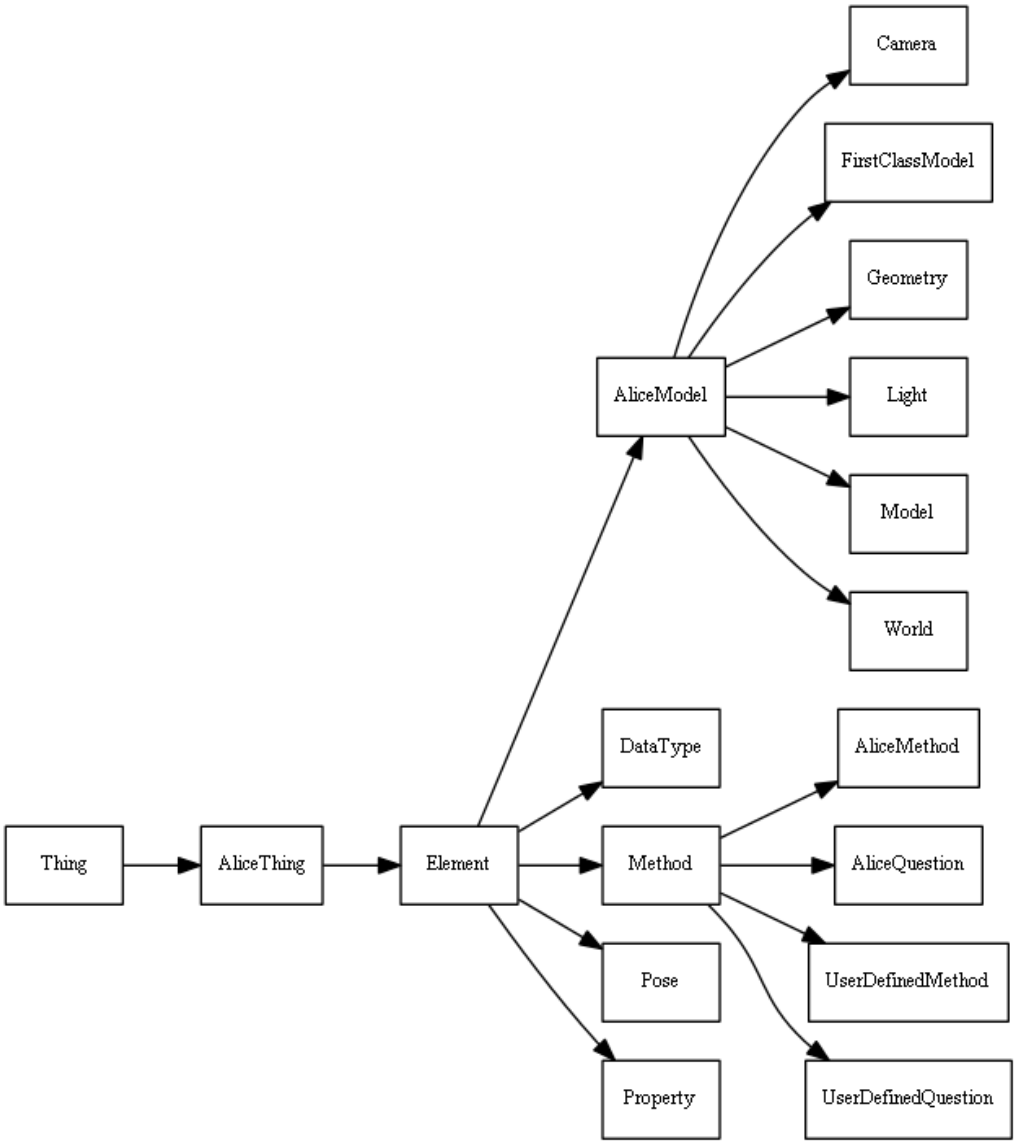

Abbildung 4.2.: Klassenhierarchie der betrachteten Ontologie

Intuitiv erscheint es logisch, dass Namen (Wortgruppen) von Individuen, die ein Modell beschreiben, eher durch Substantive geprägt werden, wohingegen ein Verb zumeist Methoden modelliert. Diese Art der Namensgebung für Methoden und Objekte (Modelle) ist eine weitverbreitete Konvention der Softwaretechnik und wird so auch von Brügge in seinem Buch „Objektorientierte Softwaretechnik“ beschrieben [BD04]. Unter dieser Annahme kann nun die Ontologie analysiert werden. Tatsächlich zeichnet sich innerhalb der *Alice*-API eine derartige Tendenz ab.

Stellvertretend für diese Beobachtung sollen die folgenden zwei Beispiele betrachtet werden: Zunächst das Ontologie-Individuum `bigFish`, welches der Super-Klasse „AliceModel“ angehört. Zieht man die zuvor engeführte Definition von Miller zu Rate, so wird offenbar, dass das Wort „fish“ (Fisch) der Kopf der Phrase und somit das relevante Wort ist. Dafür spricht, dass dieses Wort für die Wortgruppe unverzichtbar ist, wohingegen das Wort „big“ (groß) den Phrasen-Kopf nur gemäß der Definition modifiziert. Vor allem für eine spätere Verknüpfung von Text und Ontologie ist es eher wahrscheinlich, dass der Terminus „Fisch“ oder ein entsprechendes Synonym verwendet wird, um eine Szenerie zu beschreiben, als dass das zugehörige Attribut eine übergeordnete Rolle spielt. Folglich ist für dieses Modell

das Substantiv das relevante Wort. Als zweites Beispiel soll das Individuum `introduceIgloo` aus der Klasse „UserDefinedMethod“ betrachtet werden. Hier ist die Sachlage nicht ganz so eindeutig. Sowohl „introduce“ (einleiten) als auch das betrachtete Objekt „Igloo“ (Iglu) könnten als relevant gelten. Doch auch hier kann die Definition zu Rate gezogen werden. Offensichtlich handelt sich hier um eine auszuführende Aktion. Folglich ist das Wort „introduce“ (einleiten) für den Sinn der Wortgruppe unabdingbar, wohingegen das Objekt „Igloo“ die Aktion nur modifiziert, beziehungsweise in dem Fall auf das Objekt richtet. Ein weiteres Kriterium, das für diese Auswahl spricht, lässt sich in der Struktur der Ontologie erkennen: Hier wird offenkundig, dass das Modellindividuum `Igloo` mit dem Methodenindividuum `introduceIgloo` über die Relation „hasMethod“ verknüpft ist. Somit wäre eine Auswahl des Wortes „Igloo“ als relevantes Wort für das Individuum `introduceIgloo` eine unnötige Doppelung. Führt man diese Betrachtungen weiter, über mehr Individuen und andere Klassen, so ergeben sich empirisch die folgenden Zuordnungen:

UserDefinedMethod $\Longrightarrow$ Verb
AliceMethod $\Longrightarrow$ Verb
AliceModel $\Longrightarrow$ Substantiv
Pose $\Longrightarrow$ Verb
AliceQuestion $\Longrightarrow$ Substantiv
Property $\Longrightarrow$ Substantiv

Natürlich gibt es zudem den trivialen Fall, bei dem der Name des Individuums nur aus einem Wort besteht. In diesem Fall muss dies natürlich auch das relevante Wort sein.

Die ersten drei Zuweisungen wurden bereits thematisiert. Die folgenden drei sollen anhand der in Tabelle 4.2 aufgelisteten Beispiele erläutert werden: Fragen („AliceQuestions“) sind zumeist derart gestaltet, dass sie ein Modell um eine Attribut erweitern (erstes Beispiel für „AliceQuestions“ in der Tabelle) oder aber ein Modell mit einem anderen verknüpfen (zweites Beispiel für „AliceQuestions“ in der Tabelle). In beiden Fällen wird das Attribut oder die Relation vorrangig über das Substantiv bestimmt. Auch dies lässt sich auf die von Brügge proklamierten softwaretechnischen Konventionen zurückführen, welche die Beschreibung von Attributen durch Nominalphrasen fordern. Gleiches gilt für Eigenschaften („Property“-Klasse), die auf den ersten Blick auch durch Adjektive oder Adverbien beschrieben werden könnten. Dies ist auch teilweise der Fall. Eigenschaften treten jedoch zumeist als Individuen mit nur einem Wort als Name auf, wodurch wiederum der triviale Fall Anwendung findet. Sind die Namen länger, umschreiben sie die Eigenschaft und werden wiederum über Nominalphrasen modelliert. Beides kann auch in der Tabelle nachvollzogen werden. Posen ihrerseits werden intuitiv eher durch Substantive beschrieben. Tatsächlich sind es aber zumeist Verben, die sinngebend sind, was damit zusammenhängt, dass Posen Sonderformen von Methoden sind und folglich auch ähnlich formuliert werden.

Zuletzt muss in diesem Zusammenhang erörtert werden, was geschieht, sollten innerhalb einer Wortgruppe zwei oder mehr Wörter die geforderte Wortart besitzen und somit als potentielle Kandidaten für das relevante Wort gelten. Ein intuitiver Ansatz wäre, ein Maß für die Auftretenswahrscheinlichkeit zu Hilfe zu nehmen. So könnte beispielsweise für das Individuum `EskimoGirl`, das aussagekräftigere und eher in der Ausdrucksweise variierte Teilwort „girl“ (Mädchen) als relevantes Wort bestimmt werden. Im Allgemeinen ist der Informationsgehalt jedoch umgekehrt proportional zur Auftretenswahrscheinlichkeit. So treten Verbformen, die als Hilfsverben fungieren können, natürlich deutlich häufiger auf als andere Verben, tragen jedoch kaum Information mit sich. Folglich wären sie eine schlechte Wahl für das relevantes Wort. Aus diesem Grund wird in den meisten Fällen das erste Wort, welches die geforderte Wortart besitzt gewählt, da Objekt und Methoden beziehungsweise Funktionen-Namen dazu tendieren, die Hauptinformation im ersten

(Haupt-)Wort zu tragen. Die einzige Ausnahme bilden die Substantive. Hier wird eine einfache Heuristik angewendet. Der häufigste Grund für das Auftreten von zwei (und selten auch mehr) Substantiven in einer Wortgruppe ist die Zusammensetzung von Substantiven, leicht nachzuvollziehen anhand der Individuen `EskimoGirl`, `RockChair` und `Snowwoman`. Da Englisch eine germanische Sprache ist, gilt für Zusammensetzungen die „right-hand head"-Regel [Pla03], die besagt, dass das Hauptwort der Zusammensetzung (nahezu) immer am Ende der Zusammensetzung steht. Folglich wäre es richtig, immer das letzte Substantiv, das zur Verfügung steht, zu verwenden. Jedoch werden hier Wortgruppen betrachtet und nicht nur zusammengesetzte Substantive. Das letzte Substantiv der Wortgruppe könnte somit auch außerhalb der Substantiv-Komposition beispielsweise als Objekt Verwendung finden. Aus diesem Grund und der Beobachtung, dass die meisten Zusammensetzungen aus zwei Teilwörtern bestehen, sowie der zuvor getroffenen Annahme, dass die Hauptinformation zu Beginn eines Objekt- oder Methoden-Namens steht, wird schließlich im Falle der Substantive das zweite mögliche Wort, soweit vorhanden, gewählt.

Tabelle 4.2.: Exemplarische relevante Worte

Klasse (zugehörige Wortart)	Beispiel-Individuum	relevantes Wort
AliceModel (Substantiv)	bigFish LeftUpperArm	Fish Arm
UserDefinedMethod (Verb)	ExplorePenguins turn_head_right	Explore turn
AliceMethod (Verb)	stand up move to	stand move
Pose (Verb)	drive_right sitting	drive sitting
AliceQuestion (Substantiv)	<subject>'s depth <subject>distance to<object>	depth distance
Property (Substantiv)	probability of true right	probability right

Sollte unter diesen Vorgaben kein relevantes Wort gefunden werden, wird einfach ein Substantiv als sinngebendes Wort angenommen, sollte es keines geben, ein Verb. Schlägt auch das fehl, wird das erste Wort der Wortgruppe ausgewählt. Dies gilt natürlich auch für die nicht aufgeführten Klassen (siehe Klassenhierarchie unter 4.4). Für diese Klassen ist keine Einteilung möglich oder nötig, da sie entweder sehr unterschiedliche Individuen-Namen enthalten oder nur von wenigen Individuen bevölkert werden. Eine aufwendigere Ausnahmelösung ist nicht nötig, da diese Ausnahmen sehr selten sind. Weiterhin befinden sich die meisten Individuen innerhalb der betrachteten Klassen. Davon abgesehen sind weitreichende Überlegungen aufgrund der eingangs erwähnten Informationsarmut kaum möglich und dieser einfache Ansatz liefert gute empirische Ergebnisse (siehe Kapitel 6.2).

4.6. Anpassen der Wortart

Wie im vorherigen Kapitel erläutert, wäre es wünschenswert, das sinngebende Wort wäre für Methoden ein Verb, für Modelle ein Substantiv und so weiter. In Ausnahmefällen kann dies aber aus unterschiedlichen Gründen nicht zutreffen. Zum einen kann es sein, dass die Wortgruppe kein entsprechendes Wort enthält, oder aber der Name des Individuums besteht nur aus einem Wort. In diesem Fall ist eine Auswahl nicht möglich. Ist der Name zudem auch noch sinnentstellend, so wird eine spätere korrekte Synonymfindung unmöglich. Ein Beispiel für einen derartigen Fall ist die Methodenindividuum `Introduction`. Das Wort „introduction" (Einleitung) ist eindeutig ein Substantiv. Wie zuvor erörtert

sollte eine Methode aber vornehmlich durch ein Verb beschrieben werden. Außerdem finden sich für diesen Fall eher treffende Synonyme unter Verben als unter Substantiven. Aus diesem Grund wird an dieser Stelle die Wortart wie folgt angepasst: Für Individuen der Klassen „AliceMethod“ und „UserDefinedMethod“ zu Verb, für Individuen der Klasse „AliceModel“ zu Substantiv, vorausgesetzt, diese Wortart war nicht bereits zuvor gesetzt. Wie man leicht anhand des Beispiels erkennen kann, wird an dieser Stelle eine fehlerhafte Wortartmarkierung in Kauf genommen („introduction“ wird in diesem Fall als Verb markiert). Dieser Fehler ist aber gewollt und wird bei der Bestimmung der Synonyme genutzt und ausgeglichen.

4.7. Synonym-Bestimmung

Nach den vorbereitenden Arbeitsschritten können nun Synonyme für die relevanten Wörter bestimmt werden. Hierfür wird, wie im Kapitel 2 beschrieben, WordNet genutzt. Die einzelnen Verarbeitungsschritte, die im Folgenden beschrieben werden, können auch im Aktivitätsdiagramm zu diesem Kapitel (Abbildung 4.3) nachvollzogen werden. Ein Möglichkeit für die Suche in WordNet wäre, für ein bestimmtes Wort alle möglichen Synonyme bestimmen zu lassen ohne die Wortart zu betrachten. Doch gerade im Englischen sind konjugierte Verben, Substantivierungen oder Adjektive häufig gleich; die Menge der gefundenen Synonyme wäre enorm. Eine derartiges Synonymspektrum sollte aber vermieden werden, denn die Mehrzahl der so gefundenen Synonyme wäre nicht treffend[4] und müsste später zusätzlich aussortiert werden, was sowohl erhöhten Aufwand als auch gesteigerte Fehleranfälligkeit nach sich ziehen könnte. Folglich sollten Synonyme unter Vorgabe der gewünschten Wortart gesucht werden. Aus diesem Grund wurde in den vorangegangen Arbeitsschritten die Wortart für bestimmte Problemklassen festgelegt, für Methoden Verben, für Modelle Substantive und so weiter. Für die meisten Fälle lässt sich so eine Menge von mehr oder weniger treffenden Synonymen bestimmen, das heißt, nur die ersten beiden im Diagramm skizzierten Arbeitsschritte werden benötigt.

Nichtsdestotrotz sollten auch die Ausnahmefälle konzeptionell gelöst werden. Eine zu behandelnde Ausnahme wurde willentlich durch den vorherigen Arbeitsschritt erzeugt. Diese kann im Diagramm unter den Aktivitäten 3 bis 5 nachvollzogen werden. Es handelt sich um Fälle bei denen Klassenzugehörigkeit und Wortart des relevanten Wortes nicht zusammenpassen, das heißt die Wortartmarkierung explizit (mitunter absichtlich falsch) gesetzt wurde. Aus dem Beispiel des vorherigen Kapitels (siehe Abschnitt 4.6) wird leicht ersichtlich, dass bei derartigen Fällen nach einer abgeleiteten Wortform mit demselben Wortstamm gesucht werden sollte, die mit der geforderten Wortart konform ist. Genau dies ist mit WordNet möglich, indem zunächst zu dem gesuchten Wort alle abgeleiteten Wörter gesucht werden. In dieser Wortmenge wird dann nach einem Wort mit der gewünschten Wortart gesucht. Ist dies erfolgreich, so kann zu diesem Wort, anstelle des ursprünglichen Wortes, eine Synonymmenge bestimmt werden. Für das bekannte Beispiel würde dies bedeuten, dass für das Ausgangswort „introduction“ die abgeleitete Form „introduce“, mit der zugehörigen Wortart „Verb“, gefunden wird. Da dies eine Methode sein soll, wurde mit einem Verb die gewünschte Wortform gefunden; die Ausnahmebehandlung wurde erfolgreich abgeschlossen.

Sollte die Synonymmenge immer noch leer sein, muss zwangsweise nach allen möglichen Synonymen, zunächst Substantiven, gefolgt von Verben und, sollte auch dies erfolglos bleiben, ohne Berücksichtigung der Wortart gesucht werden – im Diagramm in Abbildung 4.3 unter den Ablaufschritten 6 bis 8 zu finden. Schlägt auch diese Suche fehl, ist das

[4] Ein Beispiel hierfür wäre das englische Wort „standing“, welches als Substantiv unter anderem die Bedeutung von „social or financial or professional status or reputation“ besitzt. Diese Bedeutung ist jedoch für eine Nutzung als Methode nicht relevant.

zuvor als relevant markierte Wort WordNet nicht bekannt. Dies scheint zunächst widersprüchlich, denn wie kann ein sinngebendes Wort in einem Wörterbuch nicht enthalten sein? Tatsächlich wurde das relevante Wort aber nur anhand der Wortart bestimmt. Der Wortart-Markierer, im konkreten Fall der Stanford POS-Tagger, erkennt aber im Allgemeinen mehr Wörter als WordNet. Außerdem werden unbekannte Wörter als Eigenname oder zusammengesetztes Substantiv eingestuft und somit als Substantiv markiert (siehe Kapitel 2). Der Entwurf einer geeigneten Ausnahmebehandlung ist an dieser Stelle jedoch geradezu simpel: In diesen Fällen wird einfach, wie im Diagramm in Abbildung 4.3 unter Punkt 9 bis 11 zu sehen, ein anderes als das zunächst als relevant erkannte Wort für die Synonymfindung verwendet. Dabei sollten, wie bereits als Ausnahmebehandlung bei der Bestimmung des relevanten Wortes (siehe 4.5), Substantive, gefolgt von Verben bevorzugt werden.

4.8. Sequentielle Teilwortfindung und erneute Synonymbestimmung

Trotz aller betrachteten Ausnahmen ist es auch an dieser Stelle noch möglich, dass kein Synonym gefunden werden konnte. Um diesen Fall zu veranschaulichen soll ein Beispiel-Modell-Individuum aus der betrachteten *Alice*-Welt dienen; es trägt den Namen `uglyfish`. In den bisherigen Verarbeitungsschritten wurde der Name nicht in zwei eigenständige Worte aufgeteilt, da die Binnenmajuskel-Schreibweise zum Wortanfang hier programmierseitig nicht angewendet wurde. Allerdings erkennt der Wortart-Markierer einen möglichen Eigennamen und markiert das Wort folglich als Substantiv. Da der Name nur aus einem Wort besteht, wird dieses auch das relevante Wort. Auch die Wortart muss nicht angepasst werden, da eine Substantiv-Markierung für Modelle erwünscht ist. Jedoch schlägt der nächste Verarbeitungsschritt fehl, denn WordNet findet keine Eintragung für das nicht existente Wort „uglyfish“, egal ob nach Substantiven, abgeleiteten Wortformen oder jeglichen Synonymen gesucht wird. Auch die letzte Ausnahmebehandlung, die Synonymfindung für ein anderes Wort des Namens, schlägt an dieser Stelle fehl, da der Name weiterhin nur aus einem Einzelwort besteht. Diese Art Fehler entsteht aufgrund der Nichtbeachtung softwaretechnischer Konventionen.

An dieser Stelle wäre es natürlich möglich, diese Fälle nicht weiter zu betrachten und auf Fehlverhalten in der Versuchsumgebung zu verweisen, dessen Ausgleich nicht Teil der Zielsetzung ist. Leider ist diese Art Fehler eher die Regel als die Ausnahme und eine Nichtbehandlung wäre gleichzusetzen mit einer schlechten Performanz des gesamten bisher erstellen Entwurfs. Gleichwohl ist eine Fehlerauflösung für die meisten dieser Fälle unter Einbeziehung von WordNet möglich. Die Abbildung 4.4 skizziert das Vorgehen für diese Ausnahmen. Die grundsätzliche Idee lautet dabei wie folgt: Der ursprüngliche Name wird an allen möglichen Stellen geteilt und in der Folge überprüft, ob für beide Teilwörter eine Eintragung in WordNet existiert. Ist dies der Fall wird der Name an dieser Stelle getrennt und als neuer Name für das Individuum gesetzt.

Der konkrete Ablauf unterteilt sich in vier aufeinanderfolgende Einzelschritte; diese können im Aktitätsdiagramm (Abbildung 4.4) nachvollzogen werden: Zunächst wird überprüft, ob an den eigentlichen Namen lediglich ein „l“ oder „r“, als Abkürzung für „links“ beziehungsweise „rechts“, angehängt wurde. Ist dies der Fall, wird der Name an jener Stelle getrennt (Schritt (1)). Als zweite Alternative wird der Name an allen möglichen Stellen getrennt und jeweils für die Teilworte überprüft, ob Einträge in WordNet existieren. Sollten die Anfragen erfolgreich sein wird der Name an dieser Stelle getrennt (Schritt (2)). Andernfalls wird nach einzelnen Wörtern gesucht, zunächst beginnend vom Ende des Namens, danach beginnend am Anfang (Schritte (3) und (4)). Letzteres liefert erfahrungsgemäß viele falsche positive Ergebnisse, weshalb diese Möglichkeit zuletzt ergriffen werden sollte und

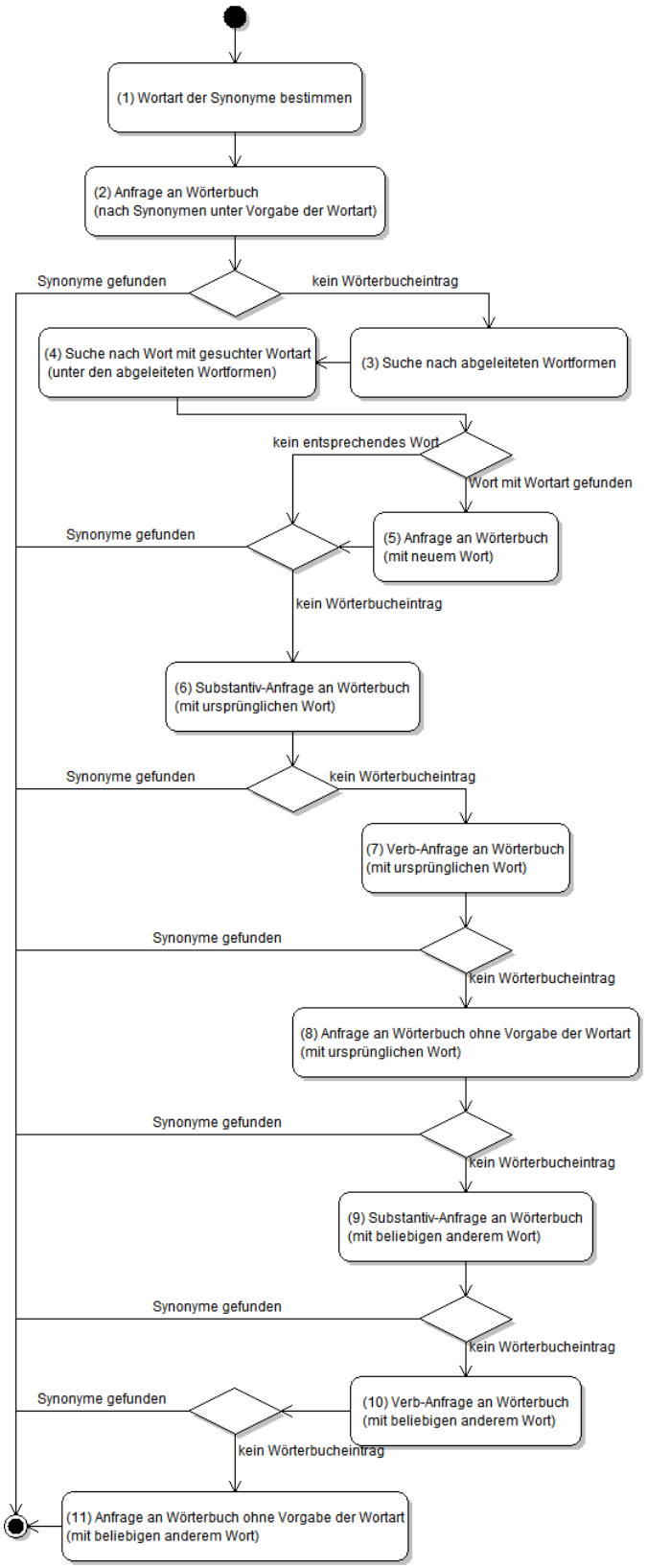

Abbildung 4.3.: Verarbeitungsablauf der Synonymbestimmung

die Doppeltwortsuche in jedem Fall zu bevorzugen ist. Beispiele für die unterschiedlichen Problem-Klassen können in der Tabelle 4.3 nachvollzogen werden.

Für das zu Beginn eingeführte Beispiel-Individuum `uglyfish` bedeutet dieses Vorgehen,

dass WordNet für die Wörter „ugly“ und „fish“ Eintragungen findet. Folglich wäre der neue Name des Individuums `ugly fish`.

Ein Algorithmus, der den Namen an mehr als einer Stelle teilt, wäre ebenso denkbar; jedoch würde dies zu einer Vielzahl von Falscherkennungen führen und das Ergebnis somit eher verfälschen als bereichern. Um sich diesen Sachverhalt klar zu machen, genügt es, sich vor Augen zu führen, dass viele Wörterbücher beziehungsweise Synonym-Datenbanken, auch WordNet, Eintragungen für einzelne Buchstaben enthalten, so zum Beispiel „g“ als Entsprechung für „Gramm“.

Tabelle 4.3.: Sequentielle Teilwortfindung

Problem-Klasse	Fehlerhafter Name	Sequentiell geteilter Name
„l“ oder „r“ angehängt	polel wheelr	pole l wheel r
Doppelwort	uglyfish snowwoman	ugly fish snow woman
Einzelwort (von hinten)	exlpore IEEERemainder	pore Remainder
Einzelwort (von vorne)	skyir handletopr	sky handle

Konnte der Name in ein oder zwei sinnvolle Teilwörter aufgeteilt werden, müssen die Schritte, die in den Kapiteln 4.2 bis 4.7 erläutert wurden, wiederholt werden, um schließlich Synonyme zu erhalten.

Sicherlich gibt es Verfahren, die bessere Ergebnisse bei der Findung von Teilwörtern liefern. Das von Enslen et. al [EHPVS09] beschriebene Verfahren zur automatischen Trennung von Bezeichnern in Quelltext erreicht eine Gesamtgenauigkeit von über 97%[5]. Die Erkennung von nicht durch Binnenmajuskel-Schreibweise getrennten Bezeichnerbestandteilen liegt jedoch nur bei gut 22%. Doch gerade diese Fälle werden hier gesondert betrachtet. Das Verfahren von Enslen ist somit hier nicht anwendbar. Ein weiterer Grund für die Favorisierung des hier beschrieben Algorithmus ist, dass das eigentliche Ziel darin besteht, Synonyme mit Hilfe von WordNet zu bestimmen. Folglich ist es sinnvoll, WordNet als Grundlage für die Auswahl der Teilwörter zu nutzen, da so garantiert ist, dass eine anschließende Synonym-Anfrage auch tatsächlich zu Ergebnissen führt.

Abschließend soll die Frage geklärt werden, warum diese Fehlerauflösung erst an dieser Stelle Anwendung findet. Denkbar wäre zum Beispiel ein Einsatz direkt nach der Namensextraktion (siehe Kapitel 4.1). Dies würde jedoch dazu führen, dass viele Wörter betrachtet werden, die später doch nicht als relevantes Wort ausgewählt werden. Ebenso könnten auf dieses Weise zu viele Teilwörter entstehen, die zwar syntaktisch aus dem Ursprungsnamen ableitbar sind, jedoch keine semantische Verbindung zu selbigen besitzen. Dies wiederum würde das Ergebnis des Wortart-Markierers und somit auch die Auswahl des relevanten Wortes beeinflussen und höchstwahrscheinlich verfälschen. Eine zweite Möglichkeit, das beschriebene Verfahren früher einzusetzen, wäre als Ausnahmefall bei der Synonym-Bestimmung (siehe Kapitel 4.7). Davon ist allerdings aufgrund der Restfehlerwahrscheinlichkeit der Namensaufteilung abzusehen. Nehmen wir exemplarisch an, die betrachtete *Alice*-Welt würde ein Individuum `fountainOfQuirinus` mit zugehöriger Super-Klasse „AliceModel“ enthalten. Dem beschriebenen Programmablauf zufolge würde zunächst das Wort „Quirinus“ betrachtet werden. Die Synonym-Anfrage schlägt

[5] Das Verfahren wäre somit auch anstelle der eigens entwickelten Namensextraktion nutzbar (siehe Kapitel 4.1 und 5.2)

aber fehlt, da WordNet zwar über ausgeprägte, jedoch unvollständige Kenntnisse der römischen Mythologie verfügt, denn der antike Gott der Quelle „Quirinus“ ist WordNet unbekannt. Folglich würde der Begriff in die Teilwörter „Quiri“ und „nus“ aufgelöst werden, denn WordNet erkennt in „nus“, die englische Pluralform des griechischen Buchstabens „Ny“ (ν). Daher würden Synonyme für den Begriff „nu“ bestimmt werden. Diese Suche wäre jedoch sinnentstellend. Folglich wäre es besser, Synonyme für ein anderes Wort der Wortgruppe, in diesem Fall „fountain“ (Brunnen) als dem zunächst als relevant markierten zu suchen. Dies ist genau das Vorgehen, das als Ausnahme bei der Synonym-Bestimmung angewendet wird.

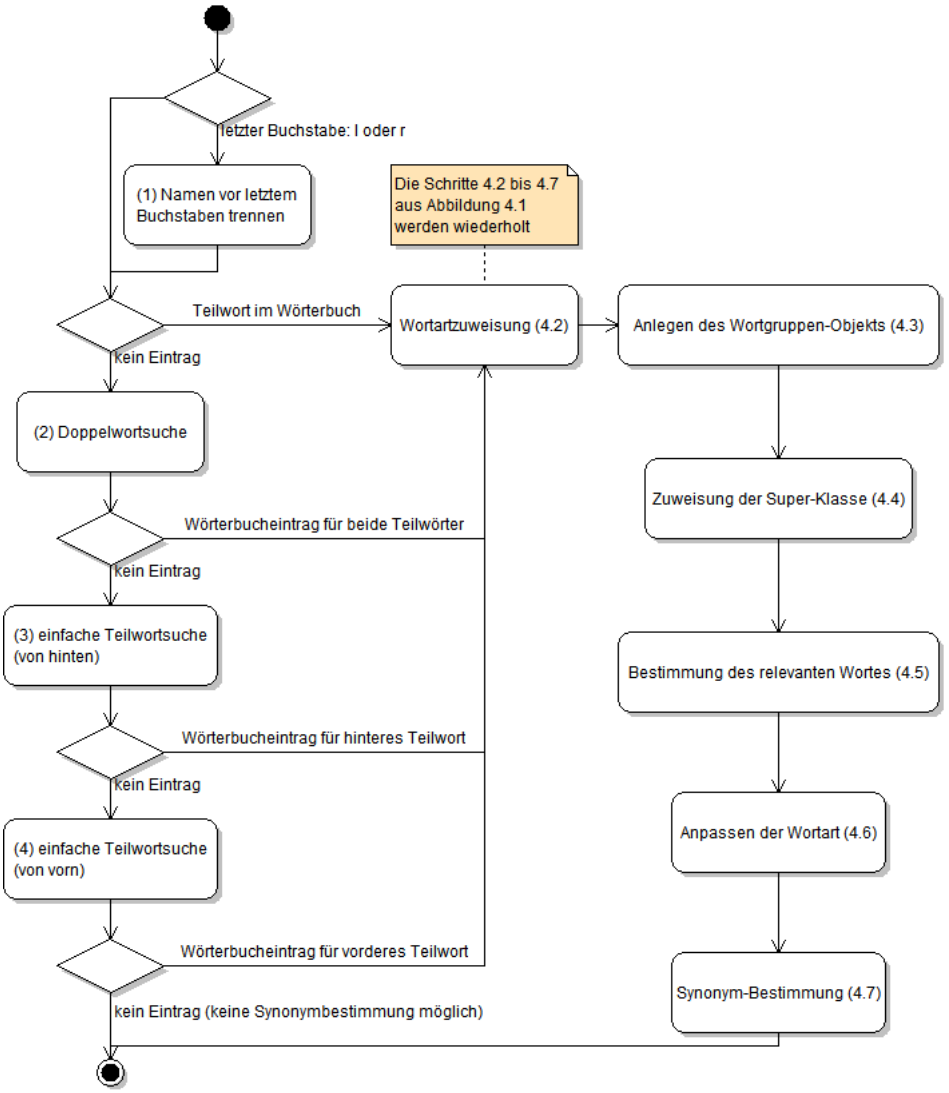

Abbildung 4.4.: Sequentielle Teilwortfindung und erneute Synonymbestimmung

4.9. Synonym-Auswahl

Die Auswahl der Synonyme beschränkt sich aus technischen Gründen auf Ontologie-Individuen der Klasse „AliceModel“. Hierfür gibt es drei Gründe.

- Da die Klasse „AliceModel“ eine Super-Klasse mehrerer Modell-Klassen ist (siehe Kapitel 4.4), wird diese von vielen Individuen bevölkert. Bei allen betrachteten Beispiel-*Alice*-Welten lag der Anteil bei mehr als der Hälfte. Folglich führt eine Auswahl der Synonyme für Individuen dieser Klasse bereits zu einer signifikanten Reduzierung der Gesamtsynonymmenge.

- Individuen dieser Klasse liefern besonders viele Synonyme. Alleine das Wort „head“ (Kopf), des Namens eines Individuum der besagten Klasse, hat im Englischen über dreißig Bedeutungen. Eine Reduzierung der Synonyme ist also an dieser Stelle besonders vorteilhaft.
- Individuen dieser Klasse liefern die besten Ansatzpunkte für eine Synonymauswahl. Hierzu werden Eigenschaften, die zuvor speziell für diese Individuen in der Ontologie definiert wurden und eine spezielle hierarchische Namensgebung dieser Individuen genutzt.

Bevor die Verfahren im einzelnen erläutert werden können müssen jedoch einige sprachwissenschaftliche Begriffe zum besseren Verständnis eingeführt werden:

Hyperonym:	Der Oberbegriff eines Begriffs wird als Hyperonym bezeichnet. So ist beispielsweise das Wort „Gemüse“ ein Hyperonym bezüglich des Wortes „Gurke“. Dreht man die Relation um, so spricht man von Hyponymen (Unterbegriffen). Hyponymie wird als Eigenschaft im folgenden jedoch nicht verwendet.
Holonym:	Holonymie beschreibt eine Teil-Ganzes-Relation, derart, dass ein Holonym-Begriff einen anderen beinhaltet. Folglich ist das Wort „Gesicht“ ein Holonym zu „Nase“.
Meronym:	Meronymie ist die Umkehrrelation der Holonymie, das heißt, ein Meronym-Begriff ist ein Teil eines anderen. So ist „Nase“ ein Meronym zu „Gesicht“ oder „Finger“ ein Meronym bezüglich „Hand“.

Diese Eigenschaften von Wörtern, die alle mithilfe von WordNet zu einem gegebenen Wort (Synset) bestimmbar sind, können genutzt werden um die Synonymmenge zu reduzieren. Hierzu werden zwei Ansätze gesucht, einer führt über die Holonyme des jeweiligen Wortes (beziehungsweise dessen Synonyme), der andere über die Meronyme. Beide Verfahren verhalten sich ähnlich, weshalb im folgenden lediglich jenes, welches Holonyme nutzt, im Detail erläutert wird.

Um unter einer Menge von Bedeutungen (Synonymen) eines Wortes diejenigen zu selektieren, die tatsächlich in den Kontext, der jeweiligen *Alice*-Welt, passen, kann die Holonym-Eigenschaft genutzt werden. Um herauszufinden, welche Bedeutung des eingangs erwähnten Wortes „head“ (Kopf) tatsälich treffend ist, kann überprüft werden, ob in der Ontologie Holonyme vorhanden sind, die mit dem Individuum `head` in Verbindung stehen. Ein passendes Holonym wäre hier zum Beispiel „body“. Es muss allerdings noch eine Verbindung zwischen den betrachteten Individuen hergestellt werden. Um die Holonym-Eigenschaft tatsächlich nutzen zu können, sollte für ein gegebenes Individuum das übergeordnete (Elter)-Individuum bekannt sein. Dies wurde bereits beim Anlegen der Ontologie durch Oleg Peters [Pet12] berücksichtigt; die Namen der Individuen sind hierfür (anders als im Kapitel 4.1 dargestellt) hierarchisch aufgebaut um die Struktur von *Alice*-Modellen wiederzugeben. Das Individuum `Head` hat tatsächlich das Suffix „Penguin~~“, lautet also `Penguin~~Head`. Ebenso gibt es ein Individuum `EskimoGirl~~Head` um die unterschiedlichen „Köpfe“ unterscheiden zu können. Ein komplexeres Beispiel für diese hierarchische Namensgebung ist das Individuum `Bunny~~chest~~head~~earL`, aus dem hervorgeht, dass das Modell `Bunny` ein Teil-Modell `Bunny~~chest` besitzt, welches wiederum einen Modell namens `Bunny~~chest~~head` beinhaltet und so weiter. Für ein gegebenes Individuum der Ontologie kann so also eindeutig das übergeordnete (Elter-)Individuum bestimmt werden.

Unter dieser Voraussetzung kann nun mithilfe der Holonymie-Eigenschaft in WordNet bestimmt werden, ob ein Synonym tatsächlich in den Kontext passt oder nicht. Das Vorgehen, das in Abbildung 4.5 veranschaulicht ist, soll nun vereinfacht dargelegt werden:

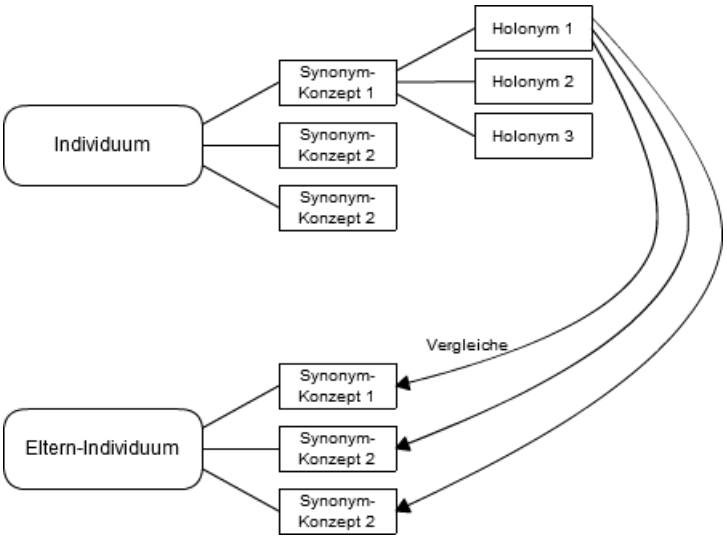

Abbildung 4.5.: Holonym-Verfahren

Zunächst werden alle Holonyme zu jeder Bedeutung (Synonym-Konzept wie in Kapitel 2 eingeführt) des momentan betrachteten Individuums mithilfe von WordNet bestimmt. Danach werden alle Synonym-Konzepte des übergeordneten (Elter-)Individuums hinzugezogen und mit den eben bestimmten Holonymen verglichen. In den meisten Fällen sollte mindestens ein Holonym mit einem Synonym des übergeordneten Individuums übereinstimmen. Dies stellt einen eindeutigen Zusammenhang zwischen den beiden Synonym-Konzepten her. Bei Übereinstimmung von Holonym des betrachteten Individuums und eines Synonyms des übergeordneten Individuums wird also das Synonym-Konzept des betrachteten Individuums als treffend angesehen und gespeichert. Sollten weitere Gleichheiten existieren, werden auch diese gespeichert. Alle anderen Synonyme des betrachteten Individuums werden als in diesem Kontext nicht zutreffend erachtet und folglich aussortiert. Sollte kein einziges Holonym einem Synonym des übergeordneten Individuums gleichen, so bleibt die Synonym-Menge unverändert.

Zum besseren Verständnis soll das Vorgehen am Beispiel-Individuum `body~~ArmL` mit dem zugehörigen übergeordneten (Elter-)Individuum `body` dargestellt werden: Bedeutungen (Synsets) des Wortes „arm“ seien

- $S_1(Ind) = \{arm\ -\ a\ human\ limb\}$ und
- $S_2(Ind) = \{arm,\ weapon,\ weapon\ system\ -\ any\ instrument\ or\ instrumentality\ used\ in\ fighting\ or\ hunting\}$.

Nun werden die zugehörigen Holonyme in WordNet angefragt. Diese sind im vorliegenden Beispiel

- $H_1(S_1(Ind)) = \{body,\ organic\ structure,\ physical\ structure - the\ entire\ structure\ of\ an\ organism\}$,
- $H_2(S_1(Ind)) = \{homo,\ man,\ human\ being,\ human - any\ living\ or\ extinct\ member\ of\ the\ family\ Hominidae\}$ und
- $H_1(S_2(Ind)) = \{weaponry,\ arms,\ implements\ of\ war,\ weapons\ system,\ munition\ -\ weapons\ considered\ collectively\}$.

Anschließend werden die zuvor bestimmten Synonym-Konzepte des übergeordneten Individuums, also `body`, mit den soeben gebildeten Holonymen verglichen. Liegt Gleichheit

vor so wird das Ausgangs-Synset des Individuums als treffend angesehen und gespeichert. Im vorliegenden Beispiel ist eine Bedeutung des Elter-Individuums `body`

- $S_n(Elter) = \{body,\ organic\ structure,\ physical\ structure - the\ entire\ structure\ of\ an\ organism\}$.

Folglich sind $S_n(Elter)$ und $H_1(S_1(Ind))$ gleich. $S_1(Ind)$ wird somit als für den Kontext treffend angesehen. Weitere Gleichheiten werden ebenfalls zur neuen Synonymmenge des Individuums hinzugefügt.

Der Algorithmus, der statt den Holonymen Meronyme nutzt, verläuft analog. Nur werden hier Individuen mit den jeweiligen untergeordneten (Kinder-)Individuen verglichen. Um den entsprechenden Zusammenhang herstellen zu könnnen wird eine zuvor in der Ontologie angelegte Relation genutzt genutzt. Modell-Individuen, die aus weiteren Teil-Modellen bestehen, besitzen eine unidirektionale Ontologie-Relation „consistsOf“ zu den jeweiligen Individuen. Für das obige Beispiel würde dies bedeuten, dass das Individuum `body` eine „consistsOf“-Relation zum Individuum `body~~ArmL` hat.

Mit diesen zwei Algorithmen können nacheinander alle unzutreffenden Synonyme aussortiert werden. Der Vorteil dieser Doppellösung besteht darin, dass bis auf das erste und letzte Element einer Individuen-Kette alle Elemente doppelt untersucht und gegebenenfalls aussortiert werden. Gleichzeitig würde bei der ausschließlichen Nutzung einer der beiden Algorithmen entweder das erste oder das letzte Element der Individuen-Kette unbeachtet bleiben.

Trotzdem erfasst das erläuterte Doppel-Verfahren nicht alle Fälle. Aus diesem Grund werden zusätzlich Hyperonyme zum Vergleich verwendet, falls entweder Holonym- oder Meronym-Anfrage erfolglos bleiben. Hierzu werden beim Holonym-Verfahren zusätzlich die Hyperonyme der Elter-Individuen gebildet und dann mit dem Ausgangsindividuum verglichen, wohingegen beim Meronym-Verfahren die Hyperonyme des betrachteten Individuums gebildet werden bevor sie mit den Kinder-Individuen verglichen werden.

Warum dieser zusätzliche Aufwand betrieben wird, soll am Beispiel des Individuums `Penguin~~Head` erläutert werden. Ein Hyperonym des Begriffs „penguin“ (Pinguin) ist „bird“ (Vogel). Eine Teil-Ganzes-Beziehung zwischen „bird“ und „head“ herzustellen ist aber wahrscheinlicher als zwischen „penguin“ und „head“.

Der gesamte Ablauf, der auch in Abbildung 4.6 nachvollzogen werden kann, ist also wie folgt: Zunächst wird das Holonym-Verfahren angewandt. Bei Erfolg werden die treffenden Synonyme gespeichert. Sollte das Verfahren erfolglos bleiben, werden die Hyperonyme der Elter-Synonyme gebildet und das Holonym-Verfahren wiederholt. Bleibt auch dies ohne Ergebnis, bleibt die Synonymmenge unverändert. In jedem Falle wird danach das Meronym-Verfahren angewandt. Wieder werden bei Misserfolg die Hyperonyme, diesmal der Individuen-Synonyme, gebildet und das Verfahren wiederholt. Werden Gleichheiten gefunden so wird eine neue Synonymmenge erzeugt beziehungsweise erweitert, andernfalls bleibt die ursprüngliche Synonymmenge erhalten.

4.10. Ontologie-Erweiterung

Abschließend soll die Ontologie um die gesammelten Informationen erweitert werden. Das bedeutet, Synonyme, der ursprüngliche Name und gegebenenfalls, wie in Kapitel 4.8 beschrieben, geteilter Name müssen in geeigneter Form gespeichert werden. Eine Möglichkeit wäre, eine neue Synonym-Klasse zu erstellen, diese mit den Synonym-Individuen zu befüllen und eine Relation zwischen ursprünglichem Individuum und Synonym-Individuum herzustellen. Neben einem unverhältnismäßigen Speicheraufwand wäre diese Variante zudem äußerst unübersichtlich. Eine zweite sich bietende Möglichkeit wäre das Hinzufügen

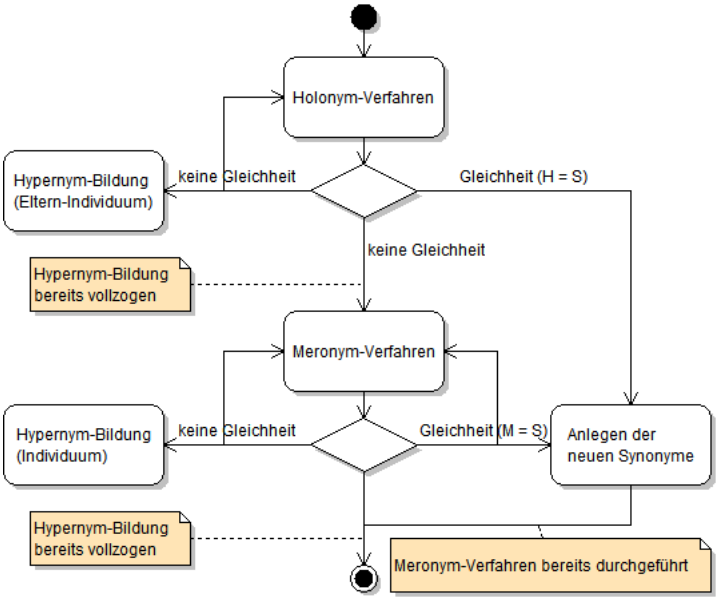

Abbildung 4.6.: Synonym-Auswahl

der Synonyme (und anderer Daten) als Attribute zu jedem Individuum. Allerdings müsste in diesem Fall für jedes Synonym ein eigenes Attribut angelegt und der Datentyp angegeben werden, was wiederum zu übermäßigem Speicherbedarf führt. Da in der vorliegenden Arbeit alle Daten in Textform vorliegen, bietet es sich an, die Synonyme (und Zusatzinformationen) einfach als Kommentar (Annotation) an das entsprechende Individuum in der OWL-Ontologie anzuhängen. Diese Lösung ist sowohl übersichtlich, gut erweiterbar als auch weniger speicherintensiv. Im Kapitel 5.8 wird zudem dargelegt, wie eine eindeutige Zuordnung des Nutztextes und Ausschluss gegenüber anderen potentiell möglichen Kommentaren gewährleistet werden.

5. Implementierung

Die wesentliche Herausforderung bei der Umsetzung des Entwurfs besteht darin, die eingesetzten Werkzeuge und Technologien in Einklang zu bringen. Das heißt, es muss ein Rahmen geschaffen werden, in dem alle Programmbestandteile kooperieren können. Die Hauptfunktionalität der Teilaufgaben erledigen die einzelnen Werkzeuge, also Wortart-Markierer, Wörterbuch und Ontologie. Diese wurden bereits im Kapitel 2 vorgestellt. Nachfolgend soll der Programmablauf vereinfacht dargelegt werden. Ebenso wird auf einige Implementierungsdetails, sowie Schwierigkeiten im Zusammenspiel der genutzten Werkzeuge hingewiesen.

5.1. Abriss des Programmablaufs

Der Ablauf des Programms unterliegt einer einfachen Struktur: Zunächst muss die OWL-Ontologie geladen und initialisiert werden. Gleiches gilt für den POS-Tagger und das WordNet-Wörterbuch. Anschließend startet die Funktion `addSynonyms`, die die eigentlichen Verarbeitungsschritte ausführt, bevor zu guter Letzt die Ontologie mit sämtlichen Erweiterungen gespeichert wird. Die Hauptfunktion `addSynonyms` ist dabei wie folgt untergliedert: Zunächst werden die in Kapitel 4.1 (Namensextraktion) bis 4.8 (Sequentielle Teilwortfindung) erläuterten Schritte für alle Individuen mittels einer `for`-Schleife durchgeführt und dabei ein Abbildung von OWL-Individuum auf zugehörige WordNet-Synsets mittels einer Java Map angelegt. Erst danach erfolgt, unter Einbeziehung der zuvor angelegten Abbildung, für alle Individuen die Synonym-Auswahl (Kapitell 4.9) und schließlich die Ontologie-Erweiterung (Kapitel 4.10). Dieses zweistufige Verfahren ist nötig, da, wie in Kapitel 4.9 beschrieben, zunächst für alle Individuen Synonyme existieren müssen um eine Auswahl treffen zu können. Einige wichtige Details der Implementierung der genannten Schritte werden im Folgenden erläutert.

5.2. Namensextraktion und -verarbeitung

Um zunächst die Individuen-Namen korrekt zu extrahieren, ist es erforderlich, die Klassenzugehörigkeit zu bestimmen, da sich im Zuge der Implementierung herausstellte, dass nicht alle Individuen, wie in Kapitel 4.1 angenommen, sprechende Namen tragen. Dies gilt allerdings nur für Individuen der Klassen „AliceMethod“ und „AliceQuestion“. Da sie von *Alice* nur intern (innerhalb der graphischen Benutzeroberfläche) genutzt werden, tragen sie Namen wie `edu.cmu.cs.stage3.alice.core.response.MoveTowardAnimation`

(Klasse: AliceMethod) statt, wie zuvor angenommen, einfach zu verarbeitende Namen wie `LeftFoot` (Klasse: Model). Allerdings ist diesen *Alice*-Objekten ein erklärendes Textfeld beigefügt, welches in der Ontologie dem entsprechenden Individuum als Kommentar angehängt ist. Für das Beispiel-Individuum enthält dieses Folgendes: „<subject>move<amount>toward<target>“. Der Inhalt des Textfeldes kann somit anstelle des nicht sprechenden Namen verwendet werden. Für alle anderen Ontologie-Klassen muss das Individuum extrahiert werden. Dieses liegt bereits in Textform vor und muss um den Ontologie-Pfad, der immer mitgeführt wird, gekürzt werden und anschließend von Binenmajuskel- in natürliche Schreibweise überführt werden. Ebenso werden sämtliche Wörter mittels der Java-Funktion `.toLower()` in Kleinschreibung transformiert. Dies ist nötig, da der Wortart-Markierer Wörter in Großschreibung nahezu ausschließlich als Substantive erkennt, was zu vielen Falscherkennungen führt. Die Tabelle 4.1 aus Kapitel 4.1 sieht also tatsächlich etwas anders aus (siehe Tabelle 5.1).

Tabelle 5.1.: Namensextraktion und Umwandlung in das natürlichsprachliche Format (an Implementierung angepasst)

Extrahierter Name	Natürlichsprachliches Format
Husky	**husky**
SitStandWalk	**sit stand walk**
body-front	body front
drive_right	drive right
Penguin2	**penguin** 2
uglyfish	uglyfish

Ansonsten arbeitet der POS-Tagger wie in Kapitel 4.2 beschrieben. Wichtig ist lediglich, das Textfeld von sämtlichen unnötigen Zeichen zu bereinigen. Hierzu gehören unnötige Leerzeichen, Sonderzeichen und Ähnliches. Zwar wären auch Wortart-Markierungen ohne diese Maßnahme möglich, doch sinkt die Genauigkeit unnötig. Ein weiteres Implementierungsdetail betrifft die Markierungen an sich. Diese werden, wie ebenfalls in Kapitel 4.2 erläutert, durch ein bestimmtes Symbol vom zu markierenden Wort getrennt. Dieses Symbol variiert allerdings zwischen verschiedenen Versionen beziehungsweise Modellen des Stanford Part-Of-Speech-Taggers. Aus diesem Grund muss das separierende Symbol dynamisch erkannt werden. Hierzu wird direkt nach der Initialisierung ein bekannter Test-Text dem POS-Tagger übergeben und das Separierungssymbol an der entsprechenden Stelle im zurückgegebenen Textfeld ausgelesen.

Die bisher gesammelten Informationen werden in zwei miteinander verknüpften Klassen, die eigens für jedes Individuum angelegt werden, gespeichert; diese Klassen können im Klassendiagramm in Abbildung 5.1 nachvollzogen werden. Wie in Kapitel 4.3 diskutiert, wären auch vorgegebene Datenstrukturen zur Speicherung möglich. Die Entscheidung für eigene Objekte fiel, um Erweiterbarkeit zu gewährleisten. So wurde im ersten Entwurf das relevante Wort noch in einer Variablen gespeichert, dann im Zuge der Implementierung in die Java-Klasse `wordCollection` integriert. Ähnliches gilt für die Funktion `getTagToWord`; statt jedes Mal nach der zugehörigen Wortart-Markierung eines Wortes zu suchen, wurde eine Funktion realisiert, die diesen Wert zurückgibt.

5.3. Bestimmung der Super-Klasse

Die Bestimmung der Super-Klasse beruht indes auf rekursiven Vergleichen der Klasse des Individuum mit einem vorgegeben Satz von gewünschten Super-Klassen (siehe Kapitel 4.4). Entspricht die Klasse einer solchen vorgegebenen Super-Klasse, so wird die Klasse als Super-Klasse des Individuums gesetzt, andernfalls wird die nächst übergeordnete Klasse

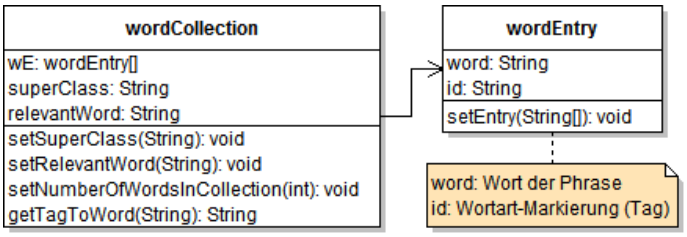

Abbildung 5.1.: wordCollection-Klasse

der aktuellen Klasse bestimmt und die Funktion damit erneut aufgerufen. Bei der Bestimmung der Klassenzugehörigkeit ergibt sich jedoch ein Problem. Wie in Kapitel 4.4 angedeutet, ist eine multiple Klassenzugehörigkeit für Individuen möglich. Gleiches gilt entsprechend für Klassen und deren übergeordneten Klassen. Aus diesem Grund liefert die OWL-API auch einen Satz von Klassen für die entsprechende Anfrage. Im Rahmen dieser Arbeit wird an dieser Stelle dann nur die erste Klasse betrachtet, da aus semantischen Gründen (siehe Kapitel 4.4) davon ausgegangen werden kann, dass Individuen eine eindeutige Klassenzugehörigkeit, beziehungsweise Klassen nur eine übergeordnete Klasse besitzen. Diese Restriktion sollte jedoch für alle in diesem Umfeld verwendeten Ontologien eingehalten werden, da an dieser Stelle ansonsten undefinierte Zustände im Programm auftreten können. Die vordefinierten Super-Klassen lauten wie folgt:

Element
AliceModel
AliceMethod
UserDefinedMethod
AliceQuestion
UserDefinedQuestion
Pose
DataType
Property

Zwei Dinge fallen beim Vergleich dieser Auswahl an Super-Klassen mit der Klassenhierarchie in Abbildung 4.2 ins Auge. Zum einen wird hier die Super-Klasse „Element“ vorgegeben, welche allen anderen Super-Klassen übergeordnet ist. Falls eine Ontologie mit anderen Klassen verwendet wird, kann so gewährleistet werden, dass zumindest irgendeine Super-Klasse gefunden wird. Da bei der Suche der Hierarchie-Baum von unten nach oben betrachtet wird – bedingt durch den rekursiven Aufruf – wird auch sichergestellt, dass, falls vorhanden, eine der anderen Super-Klassen zuerst gefunden wird. Die zweite Auffälligkeit ist das Einzelauftreten der Super-Klassen „AliceMethod“, „UserDefinedMethod“, „AliceQuestion“ und „UserDefinedQuestion“. Anschaulich wäre es denkbar, diese in der Super-Klasse „Method“ zusammenzufassen, schließlich steht diese auf der gleichen Hierarchieebene wie die Super-Klasse „AliceModel“, die ihrerseits die darunterliegenden Klassen zusammenfasst (siehe Abbildung 4.2). Allerdings müssen die vier zuvor genannten Klassen unterschiedlich behandelt werden. Zum einen werden bei Individuen der Klassen „AliceMethod“ und „AliceQuestion“, wie zu Beginn des Kapitels erläutert, nicht die Methoden-Namen, sondern die angehängten Textfelder betrachtet. Zum Anderen wird später bei Fragen für das relevante Wort die Wortart „Substantiv“ vorgegeben, bei Methoden die Wortart „Verb“ (siehe Kapitel 4.4). Aus diesen Gründen können diese Klassen nicht unter einer Super-Klasse zusammengefasst werden.

Quelltextausschnitt 5.1: Anpassen der Wortart

```
for(i = 0; i < wC.wordCount; i++){
        if(wC.wE[i].word.equalsIgnoreCase(wC.relevantWord)){
                if(wC.superClass.equalsIgnoreCase("AliceModel") && !wC.
                    wE[i].id.contains("NN")){
                        wC.wE[i].id = "NN";
                }
        }
}
```

5.4. Bestimmung des relevanten Wortes

Die Entscheidung für das relevante Wort erfolgt, wie in Kapitel 4.5 erläutert, anhand der für entsprechende Super-Klasse geforderte Wortart. Hierzu wird das `id`-Feld (siehe Abbildung5.1) jedes Wortes der betrachteten Wortgruppe (gespeichert in der `wordCollection`) überprüft und das gewünschte Wort ausgewählt. Im Falle eines Modells ist dies das zweite Wort, welches die Markierung „NN“ hat, im Falle einer Methode, das erste Wort mit der Markierung „VB“ und so weiter.

Allerdings ergeben sich zwangsläufig zwei Abweichungen von dem in Kapitel 4.5 beschriebenen Verfahren, die implementierungstechnische Gründe haben. Zum einen wird zusätzlich zur Suche nach Wörtern mit der Markierung „NN“ bei Modellen, sollte dies erfolglos bleiben, nach Wörtern mit der Markierung „JJ“ gesucht, was für „Adjektiv“ steht. Dies ist nötig, da seitens des Stanford-POS-Taggers häufig Substantive fälschlich als Adjektive markiert werden. Zum anderen sei darauf hingewiesen, dass für den Vergleich mit dem „id“-Feld die Java-String-Funktion `.contains(String)` verwendet wird, da Markierungen Erweiterungen besitzen können, um zum Beispiel Pluralformen bei Substantiven oder Beugung bei Verben anzuzeigen. So ist die Markierung für Substantive im Plural „NNS“. Für die vollständige Liste der Markierungen des Stanford POS-Taggers siehe Anhang A. Die Verwendung von `.contains(String)` gewährleistet die Konformität mit diesen Sonderformen. Die Ausnahme bildet die Verb-Suche. Hier wird nach einem exakten Treffer der Markierung „VB“ gesucht, folglich die Java-String-Funktion `.equals(String)` genutzt. Wieder kann dieses Vorgehen über Ungenauigkeiten des POS-Taggers begründet werden. Vor allem Substantive mit „ing“-Endung (Gerundium) werden häufig falsch als Verb in Progressiv/Continuous-Form erkannt. Ähnlich verhält es sich mit von Verben abgeleiteten Adjektiven, die auf „ed“ enden und fälschlicherweise als Verb markiert werden, weil die Verbform in Vergangenheit syntaktisch gleich ist.

Zusätzlich zu dieser Implementierung wurde eine zweite realisiert, die die Auftretenswahrscheinlichkeit der Kandidaten für das relevante Wort (bei mehreren Wörtern der gesuchten Wortart innerhalb der Wortgruppe) zur Auswahl nutzt. Diese liefert jedoch aus den in Kapitel 4.5 dargelegten Gründen keine guten Ergebnisse und wird deshalb bis auf weiteres nicht verwendet.

In Einzelfällen kann es vorkommen, dass die Auswahl des relevanten Wortes nicht auf die korrekte Wortart zurückgeführt werden kann. Diese Problematik wurde in den Kapiteln 4.5 und 4.6 erläutert. In diesen Fällen muss die Wortart des relevanten Wortes noch angepasst werden; die Wortart wird der durch die Super-Klasse geforderten angeglichen (siehe Kapitel 4.4) und 4.6). Innerhalb der Implementierung wird dazu geprüft, ob die entsprechende Wortart bereits gesetzt ist. Ist dies nicht der Fall, wird einfach direkt in das `id`-Feld des Wortgruppen-Objekts geschrieben (siehe Abbildung 5.1). Für Individuen der Klasse „AliceModel“ lautet der Aufruf wie in Quelltextausschnitt 5.1 gezeigt.

Gleichzeitig bereinigt dieses Vorgehen weitere durch den Wortart-Markierer bedingte Ungenauigkeiten: Der Name des Individuums `jumping` kann beispielsweise sowohl ein Verb als auch ein Substantiv sein. Mit dem Wissen, dass das Individuum zur Klasse „UserDefinedMethod“ gehört, ist aber die Verwendung als Verb deutlich wahrscheinlicher. Diese nicht unüblichen Fälle werden durch das beschriebene Verfahren ebenfalls erfasst.

5.5. Synonym-Anfrage

Für eine Synonym-Anfrage an WordNet mit vorgegebener Wortart muss zunächst der Synset-Typ gesetzt werden. WordNet unterscheidet zwischen den Typen „NounSynset“, „VerbSynset“, „AdjectiveSynset“ und „AdverbSynset“. Der Typ wird anhand der Wortart des relevanten Wortes bestimmt, also für Substantive „NounSynset“, für Verben „VerbSynset“ und so weiter. Im Normalfall ist eine Anfrage an WordNet mit diesen Parametern erfolgreich und lautet wie im Quelltextausschnitt 5.2 gezeigt.

In Kapitel 4.7 wurde bereits erläutert, wie mit diversen Ausnahmefällen umgegangen wird, was also geschieht, sollte auf die soeben beschriebene Weise kein Synset gefunden werden. Für den ersten Ausnahmefall, der die absichtlich falsch gesetzten Wortarten behandelt (siehe Abbildung 4.3 Schritte 3 bis 5), sei der Quelltextausschnitt 5.3 dargelegt.

Im Allgemeinen liefert einen Synonym-Anfrage an WordNet mehr als ein Synset, da zu einem Wort mehrere Bedeutungen gehören können. Folglich gibt WortNet immer ein Array (Feld) von Synsets zurück. Diese Synset-Feld wird im weiteren Programmablauf betrachtet und nicht treffende Synsets gegebenenfalls aussortiert.

5.6. Bestimmung von Teilwörtern

Ist das Synset-Feld jedoch noch leer, kann über die in Kapitel 4.8 beschriebene sequentielle Teilwortfindung versucht werden, den Individuen-Namen in Teilwörter aufzulösen, die WordNet bekannt sind. Ist dies möglich können über diesen Umweg doch noch Synonyme bestimmt werden. Der Quelltextausschnitt 5.4 zeigt, wie die Auflösung in zwei Teilwörter vorgenommen wird.

Da der Quelltext an dieser Stelle etwas unübersichtlich ist, soll der Ablauf der „for“-Schleife an dem bereits in Kapitel 4.8 eingeführten Beispielindividuum `uglyfish` nachvollzogen werden. Der Name wird beginnend von hinten an allen möglichen Stellen getrennt bis für beide Teilwörter ein Eintrag in WordNet gefunden werden kann. Die Pfeile im Beispiel symbolisieren dabei die WordNet-Anfrage, jeweils für das linke und rechte Teilwort. Das $\times$ zeigt dabei an, dass kein Eintrag gefunden werden konnte, das $\surd$ hingegen vermeldet Erfolg.

0		uglyfish	
1	$\times \Longleftarrow$	uglyfis h	$\Longrightarrow \surd$
2	$\times \Longleftarrow$	uglyfi sh	$\Longrightarrow \times$
3	$\times \Longleftarrow$	uglyf ish	$\Longrightarrow \times$
4	$\surd \Longleftarrow$	**ugly fish**	$\Longrightarrow \surd$

Quelltextausschnitt 5.2: Synset-Anfrage an WordNet

```
synSets = dataBase.getSynsets(wC.relevantWord,type);
//type gibt die Wortart an: "NOUN", "VERB", "ADVERB" oder "ADJECTIVE"
```

Quelltextausschnitt 5.3: Ausnahmebehandlung „falsche Wortart“

```
if(synSets.length == 0){
        // kein Synset gefunden
        synSets = dataBase.getSynsets(wC.relevantWord);
        // alle Synsets ohne Betrachtung der Wortart
        for(i = 0; i < synSets.length; i++){
                derWordForms = synSets[i].getDerivationallyRelatedForms
                    (wC.relevantWord);
                // abgeleitete Wortformen
                for(j = 0; j < derWordForms.length; j++){
                        derSynSets = dataBase.getSynsets(derWordForms[j
                            ].getWordForm());
                        // Synsets der abgeleiteten Wortformen
                        for(k = 0; k < derSynSets.length; k++){
                                if(derSynSets[k].getType().equals(type)
                                    ){
                                        synSets = derSynSets;
                                        // wenn ein abgeleitetes Synset
                                            dem erforderlichen Typ
                                            entspricht, setzte dieses
                                            als Ergebnis-Synset
                                }
                        }
                }
        }
}
```

Konnte die sequentielle Teilwortfindung abgeschlossen werden, werden alle anderen Ablaufschritte bis hin zur Synonym-Bestimmung wiederholt. Bis hierhin wurden alle Möglichkeiten zur Synonym-Bestimmung ausgereizt. Nur in Ausnahmefällen können keine Synonyme zugeordnet werden. Abschließend wird in die anfangs erwähnte Java Map das Individuum und die zugehörigen Synsets eingetragen.

5.7. Reduktion der Synonymmenge

Mithilfe dieser Abbildung kann nun die Synonym-Auswahl vorgenommen werden. Hierzu wird wieder jedes Individuum der Ontologie betrachtet; die in Kapitel 4.9 beschriebenen Verfahren werden angewendet; dementsprechend wird das Elter-Individuum beziehungsweise das Kind-Individuum geladen. Das Elter-Individuum wurde bereits bei der Namensextraktion mit aus dem hierarchischen Namen extrahiert (siehe Kapitel 4.9) und später in der `wordCollection` gespeichert. Das Kind-Individuum (oder die Kinder-Individuen) kann über die „consistsOf“-Relation in der Ontologie bestimmt werden. Mit diesen Elementen als Schlüssel können die zugehörigen Synsets aus der Java Map entnommen werden. Die in Kapitel 4.9 beschriebenen Verfahren weichen jedoch in der Implementierung in drei Punkten leicht ab; dies hängt mit der Struktur von WordNet zusammen.

Rekursivität von Holonym- und Meronym-Verfahren: Sowohl das Holonym-Verfahren als auch das Meronym-Verfahren werden rekursiv wiederholt, sollte im ersten Versuch keine Gleichheit erzielt werden (siehe hierzu Quelltextausschnitt 5.5). Diese Rekursivität ist nötig, da Meronymie und Hyponymie nicht immer unmittelbar sind. So sollte das Verfahren natürlich erkennen, dass der Begriff „head“ (Kopf) ein Holonym zu „nose“ (Nase) darstellt. Tatsächlich besitzt „nose“ in WordNet aber nur (neben weiteren sinnfremden Begriffen) das Holonym „face“ (Gesicht). Dieses wiederum besitzt das Holonym „head“. Ein Zusammenhang zwischen „nose“ und „head“ konnte also so doch noch hergestellt werden, jedoch nur

Quelltextausschnitt 5.4: Sequentielle Teilwortfindung

```
for(i = 0; i < current.length(); i++){
        if(i > 1 && dataBase.getSynsets(current.substring(0, current.
            length()-(i+1))).length != 0 &&
        dataBase.getSynsets(current.substring(current.length()-(i+1),
            current.length())).length != 0){
                returnStringBuffer.append(current.substring(0, current.
                    length()-(i+1)));
                returnStringBuffer.append('␣');
                returnStringBuffer.append(current.substring(current.
                    length()-(i+1), current.length()));
                return returnStringBuffer.toString();
        }
}
```

Quelltextausschnitt 5.5: Rekursiver Aufruf der Holonym-Funktion

```
if (returnBoolValue == false && parSynsets != null) {
        // sollte das Verfahren fehlschlagen...
        for (j = 0; j < PartHolonyms.length; j++) {
                // ...wiederhole ich es mit den Holonymen der Holonymen
                returnBoolValue = selectSynsetbyHolonyms(PartHolonyms[j
                    ],parSynsets);
                if(returnBoolValue){
                        break;
                }
        }
}
```

über Bildung von Holonymen der Holonyme (zweistufige Holonymie) des Begriffes „nose“. Derartige Beispiele, bei denen intuitiv eine Teil-Ganzes-Beziehung besteht, diese aber in WordNet nicht direkt vorhanden ist, gibt es viele, auch über mehr als zwei Stufen, was die genutzte Rekursivität rechtfertigt.

Holonym-Bildung auf Vergleichseite: Wie soeben beschrieben, sind sowohl Holonym- als auch Meronym-Verfahren rekursiv implementiert. Diese Rekursivität beschränkt sich aber auf eine Seite; die zu vergleichende Seite, im obigen Beispiel „head“, bleibt gleich. Es kann jedoch vorkommen, dass trotz der Rekursivität kein Zusammenhang zwischen zwei Begriffen hergestellt werden kann, obwohl Kontext-Gleichheit vorliegt. Auch hier sei die Problematik anhand eines Beispiels verdeutlicht (siehe Abbildung 5.2): Die Begriffe „foot“ (Fuß) und „chest“ (Brust, Brustkorb, Thorax) stehen in keinem unmittelbaren Zusammenhang, jedoch können sie unter dem gemeinsamen Kontext „body“ (Körper) zusammengefasst werden. Genau dies kann erreicht werden, erlaubt man zusätzlich zur Rekursivität auf Objektseite Holonym-Bildung auf Vergleichsseite. Im vorliegen Beispiel wäre ein direktes Holonym zu „chest“ der Begriff „body“. Auch für den Begriff „foot“ kann rekursiv „body“ als Holonym festgestellt werden; der gewünschte Zusammenhang wurde also hergestellt.

Die Holonym-Bildung auf Vergleichsseite wird bis zu drei mal wiederholt, jedoch keinesfalls rekursiv durchgeführt. Rekursivität, sowohl auf Objekt- als auch auf Vergleichsseite, würde unweigerlich in nahezu allen Fällen zu falschen positiven Erkennungen führen, da Holonyme so lang gebildet werden würden bis auf einer sehr allgemeinen Objektebene Gleichheit herrschen würde. So kann sowohl für das Wort „leaf“ (Blatt) als auch für das Wort „foot“ (Fuß) rekursiv das Holonym „organism“ (Organismus) festgestellt werden, obwohl die beiden Begriffe nicht unbedingt dem gleichen Konzept angehören.

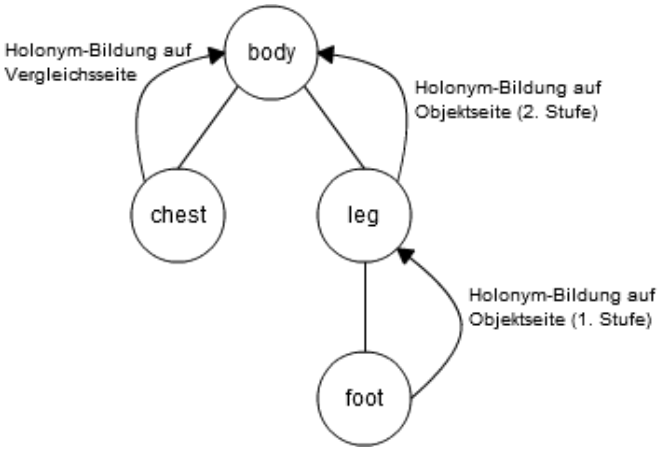

Abbildung 5.2.: Holonym-Bildung auf Vergleichseite

Erweiterte Hyperonym-Bildung: Auch die Hyperonym-Bildung wird ähnlich erweitert. In Kapitel 4.9 wurde erläutert, dass WordNet keinen direkten Zusammenhang zwischen den Begriffen „penguin“ (Pinguin) und „head“ (Kopf) herstellen kann. Aus diesem Grund werden auf Vergleichsseite Hyperonyme gebildet. Diese Idee funktioniert jedoch nur theoretisch, denn das direkte Hyperonym zu „penguin“ ist tatsächlich „sphenisciform seabird“ (flugunfähiger Kaltwasser-Seevogel) und nicht wie erwartet „bird“(Vogel). Folglich kann WordNet auch hier keinen Zusammenhang zu „head“ herstellen, ebensowenig auf der nächsten Hyperonym-Stufe „seabird“ (Seevogel). Erst wenn man erneut das Hyperonym bildet erreicht man den Begriff „bird“ und WordNet kann die gewünschte Teil-Ganzes-Beziehung zu „head“ herstellen. Aus den bereits zuvor dargelegten Gründen wird auch diese Maßnahme nicht rekursiv wiederholt. Allerdings werden Hyperonyme bis zu viermal in Folge gebildet, da die Struktur WordNet bezüglich Hyperonymie sehr feingranular ist.

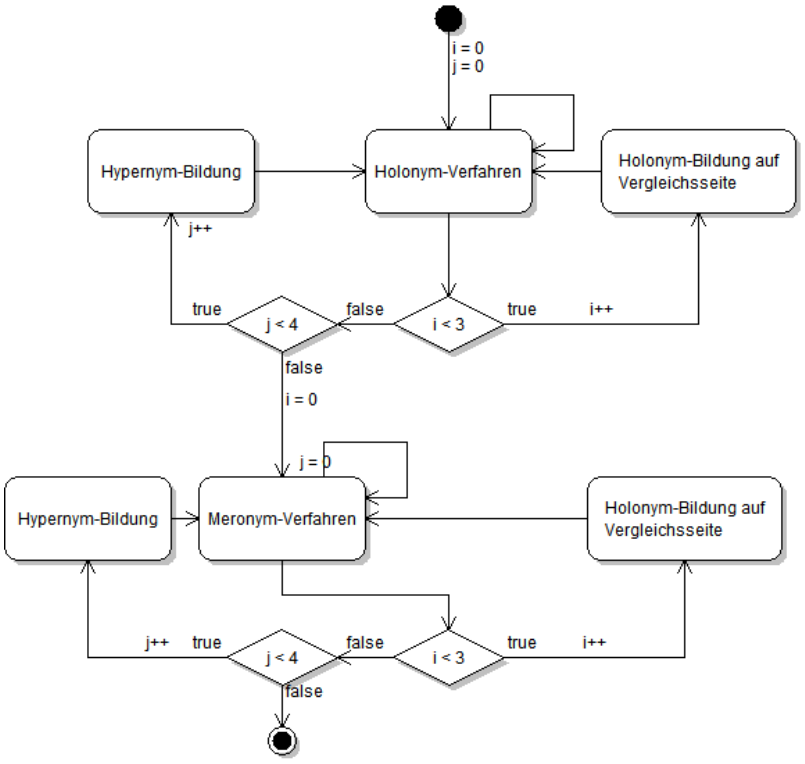

Abbildung 5.3.: Vereinfachter Programmablauf: Synonym-Auswahl

Der vereinfachte Programmablauf, wie in Abbildung 5.3 zu sehen, ist demnach wie folgt: Zunächst wird das Holonym-Verfahren rekursiv durchgeführt. Danach werden die Holonyme auf Vergleichsseite gebildet und wiederum das Holonym-Verfahren durchgeführt. Dies wird bis zu dreimal wiederholt. Anschließend werden die Hyperonyme auf Vergleichsseite gebildet und wiederum das Holonym-Verfahren angewendet. Dieser Vorgang wiederholt sich bis zu viermal. Danach wird das gesamte Vorgehen für das Meronym-Verfahren durchgeführt. Werden Gleichheiten festgestellt, so wird das betrachtete Synset als korrekt angesehen und in einem Puffer gespeichert. Sollten mehrere korrekte Synsets gefunden werden, so werden diese ebenfalls im Puffer gespeichert. Ist das Puffer-Objekt am Ende der Synonym-Auswahl nicht leer, so wird dessen Inhalt zur neuen Synset-Menge des Individuums. Andernfalls bleiben die ursprünglichen Synsets erhalten.

5.8. Speichern der Informationen

Die ermittelten Informationen müssen nun in der OWL-Ontologie hinterlegt werden: die gefundenen Synsets, der ursprüngliche Name des Individuums und, falls vorhanden, die sequentiell gefundenen Teilwörter. In Kapitel 4.10 wurden die verschiedenen Möglichkeiten bereits diskutiert und die Speicherung in einem Kommentar-Feld als die voraussichtlich beste bewertet.

Kommentare sind in OWL zur Mitführung von Metadaten vorgesehen und können an alle anderen Ontologie-Objekte angehängt werden; dadurch bleibt das hier vorgestellte Verfahren flexibel. Ebenso bestehen, abgesehen vom Speicherbedarf, keine Beschränkungen bezüglich Anzahl oder Länge der Kommentare. Allerdings müssen Kommentare untereinander unterscheidbar bleiben. Unter anderem wurde in Kapitels 5.1 beschrieben, dass auch die erklärenden Textfelder für Individuen der Klassen „AliceMethod“ und „AliceQuestion“ als Kommentare in der Ontologie gespeichert sind. Ebenso ist es möglich, dass Kommentare in der Ontologie verwendet wurden um die Klassen oder Individuen genauer zu beschreiben; eine Unterscheidung zu den hier verwendeten „Nutzdatenkommentaren“ ist also zwingend nötig. OWL bietet hierzu sogenannte „Language-Tags“ an. Diese waren seitens der OWL-Entwickler ursprünglich dafür gedacht, für eine Annotation die verwendete Sprache, zum Beispiel „en“ für Englisch, anzugeben, um mehrsprachige Dokumentation einer Ontologie zu ermöglichen. Allerdings ist das Tag-Feld frei belegbar und somit optimal geeignet, um verschiedene Kommentartypen zu unterscheiden. So wird im hiesigen Umfeld für Synsets der Tag „syn“, für den ursprünglichen Namen „name“ und für die sequentiell gefundenen Teilwörter „rescued“ verwendet. Hierdurch können die Kommentare später untereinander und auch von anderen Kommentaren unterschieden werden. Bevor die Synsets jedoch als Kommentar angefügt werden können, müssen sie in eine geeignete Form gebracht werden. Hierzu werden alle Einträge des Synset-Feldes, durch Kommata getrennt, nacheinander in ein Textfeld-Puffer (StringBuffer) geschrieben. Dieser Puffer wird anschließend in ein normales Textfeld (String) umgewandelt. Wie die Kommentare in der Ontologie gespeichert werden, kann im Quelltextausschnitt 5.6 nachvollzogen werden.

Das Verfahren für den ursprünglichen Namen und gegebenenfalls gefundene sequentielle Teilwörter verläuft analog. Abbildung 5.4 zeigt eine Ontologie innerhalb der graphischen Oberfläche des Ontologie-Betrachtungs-Werkzeugs Protegé, wobei der rote Rahmen die eingefügten Kommentare für das Individuum `uglyfish` zeigt. In Abbildung 5.5 ist eines der Kommentare in XML-Schreibweise zu sehen.

Abschließend wird die Ontologie nur noch abgespeichert entweder als neue Ontologie, oder die eingelesene wird überschrieben.

Quelltextausschnitt 5.6: Anfügen der Kommentare

```
OWLAnnotation commentAnno_Syn = df.getOWLAnnotation(
df.getOWLAnnotationProperty(OWLRDFVocabulary.RDFS_COMMENT.getIRI()),
df.getOWLLiteral(synSetString.toString(), "syn"));
// erzeuge ein Annotationsobject mit den Synsets als Inhalt und "syn"
    als Tag

OWLAxiom ax_syn = df.getOWLAnnotationAssertionAxiom(ind.getIRI(),
    commentAnno_Syn);
// erweitere die Ontologie um dieses Axiom (Annotation)

manager.applyChange(new AddAxiom(argOntology, ax_syn));
// teile dem Ontologie-Manager die Modifikation mit
```

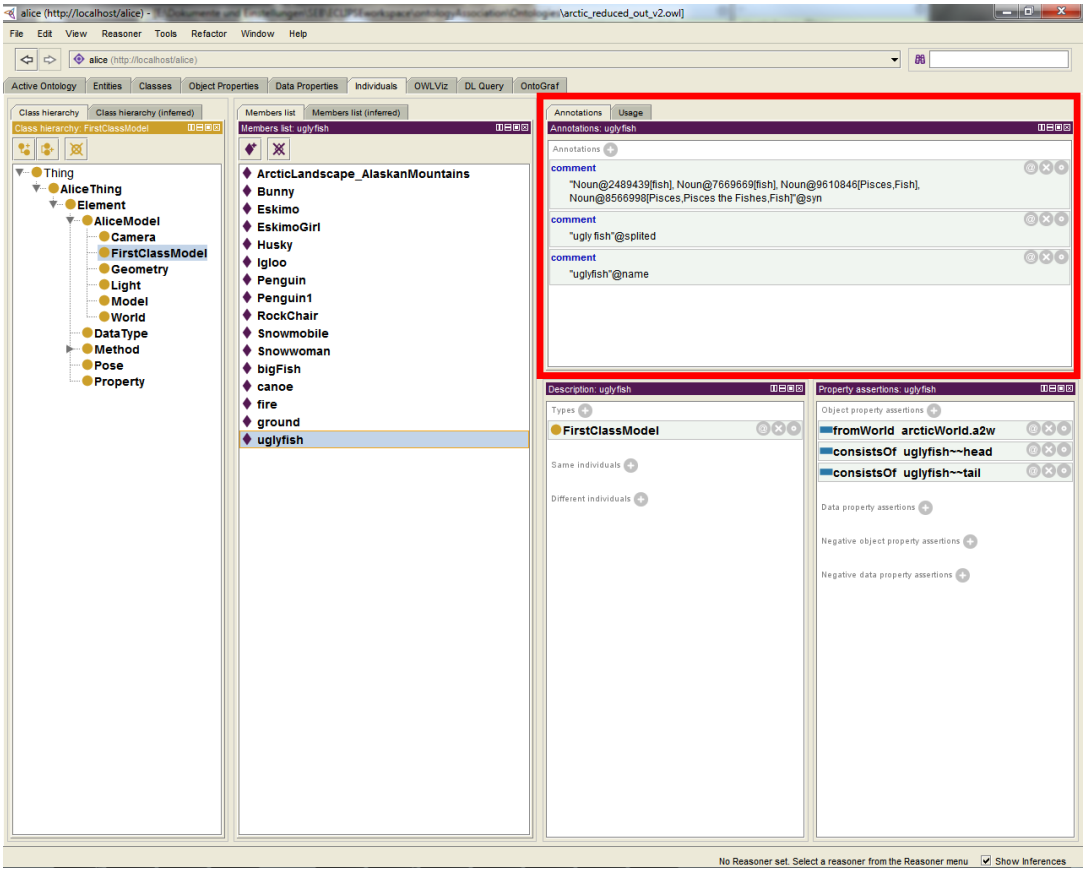

Abbildung 5.4.: Annotation in einer OWL-Ontologie (Protegé)

Abbildung 5.5.: Annotation in einer OWL-Ontologie (XML)

6. Evaluation

Die Funktionalität des erstellen Systems kann anhand von Daten, die während des Ablaufs gesammelt werden, überprüft werden, um zu zeigen, dass der in Kapitel 4 entstandene Entwurf die gewünschten Ergebnisse erzielt. Aufgrund fehlender Vergleichsarbeiten können diese Ergebnisse aber nicht anderen gegenübergestellt werden. Folglich kann an dieser Stelle ausschließlich eine intrinsische Evaluation erfolgen. Die Bewertung des Gesamtergebnis steht hierbei im Vordergrund. Darüber hinaus werden wichtige Zwischenergebnisse markanter Programmabschnitte beurteilt.

Für die Evaluation wurden vier Ontologien gewählt, die aus vier unterschiedlichen *Alice*-Welten durch den, von Oleg Peters [Pet12] entwickelten, Extraktor entstanden sind. Die zugrundeliegenden Welten sind kurze Videosequenzen mit unterschiedlichem Inhalt. Die Anzahl der verwendeten Modelle und Methoden sowie die Länge der Videos schwankt leicht, was sich in der Anzahl der Individuen der zugehörigen Ontologien niederschlägt. Im Folgenden seien die zugehörigen Ontologien mit $O(W_1)$, $O(W_2)$, $O(W_3)$ und $O(W_4)$ benannt. Zusätzlich werden die Ergebnisse mit einer Ontologie verglichen, die aus zwölf Einzelwelten extrahiert wurde, benannt mit $O(W_{sum})$. $O(W_1)$, $O(W_2)$, $O(W_3)$ und $O(W_4)$ sind dabei in $O(W_{sum})$ enthalten. Die Anzahl der Individuen der jeweiligen Ontologie sowie die zugehörige *Alice*-Welt kann Tabelle 6.1 entnommen werden.

Tabelle 6.1.: Gesamtzahl der Individuen pro Ontologie

Ontologie	# Individuen	Name der zugehörigen *Alice*-Welt
$O(W_1)$	366	`arctic_reduced.a2w`
$O(W_2)$	482	`pjsDream.a2w`
$O(W_3)$	274	`TerenceWashingtonDance.a2w`
$O(W_4)$	273	`Halloween.a2w`
$O(W_{sum})$	2341	`some_worlds.a2w`

Im Zuge der Evaluation soll zunächst betrachtet werden, wie hoch die Synonym-Abdeckung ist, das heißt, es wird geprüft, wie vielen Individuen mindestens ein Synset zugeordnet werden kann. In einem zweiten Evaluationsschritt wird die Auswahl des relevanten Wortes untersucht. Hierbei soll an ausgewählten Beispielen untersucht werden, inwieweit richtig entschieden wurde. Zuletzt wird die Leistungsfähigkeit der Synonym-Auswahl überprüft, wobei zum einen Maßzahlen für die Synonym-Reduktion betrachtet werden. Zum anderen soll wiederum an ausgewählten Beispielen gezeigt werden, dass die Auswahl sinnhaft ist, das heißt nur für den Kontext sinnfremde Synsets aussortiert werden.

6.1. Abdeckung

Eine grundlegende Maßzahl zur Bemessung der Leistungsfähigkeit des vorgestellten Systems ist die Synonym- bzw. Synset-Abdeckung. Es wird also geprüft, wie vielen Individuen einer jeden betrachteten *Alice*-Welt mindestens ein Synset zugeordnet werden kann. Das Diagramm in Abbildung 6.1 zeigt die Ergebnisse dieses Tests.

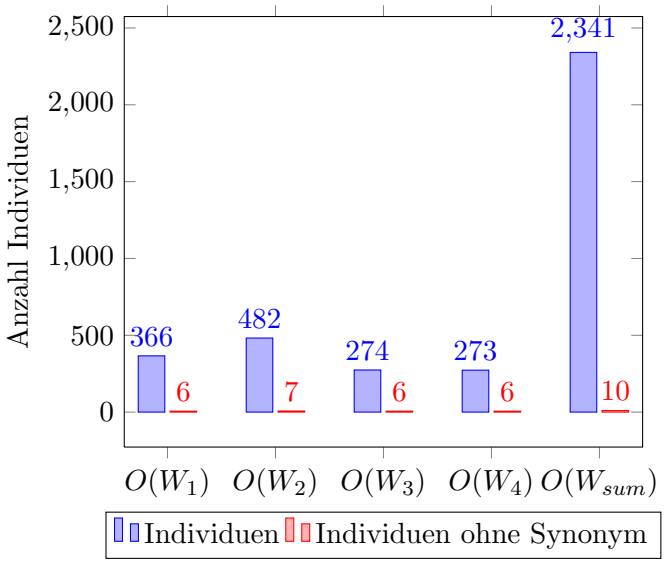

Abbildung 6.1.: Absolute Synonym-Abdeckung

Wie zu sehen ist, können für die Ontologien der Einzel-Welten $O(W_1)$, $O(W_3)$ und $O(W_4)$) jeweils sechs Individuen kein Synset zugeordnet werden, für die Ontologie $O(W_2)$ sind es sieben. Für die Ontologie $O(W_{sum})$ kann für zehn Individuen kein Synset ermittelt werden. Die Menge der Individuen, die kein Synset erhalten, bleibt indes gegenüber der Gesamtzahl der Ontologie-Individuen sehr gering. Deutlicher wird dieser Fakt anhand der relativen Abdeckung, nachzuverfolgen im Diagramm in Abbildung 6.2.

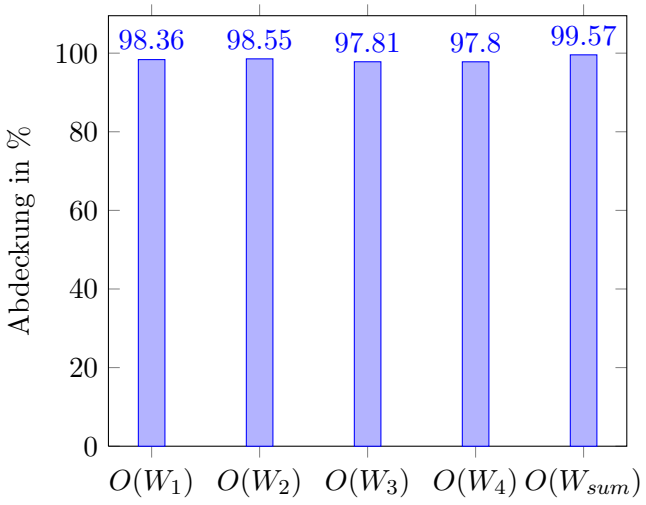

Abbildung 6.2.: Relative Synonym-Abdeckung (normale Ordinate)

Das Verhältnis von Individuen, für die kein Synset bestimmt werden konnte, gegenüber der Gesamtzahl an Individuen der Stichprobe verdeutlicht die hohe prozentuale Abdeckung. Für die relativ individuenarmen Ontologien der Einzel-Welten ($O(W_1)$ bis $O(W_4)$) konnte ein Wert von ca. 98% erreicht werden, für die deutlich größere Ontologie $O(W_{sum})$ liegt

dieser sogar bei 99.57%. Abbildung 6.3 zeigt die gleichen Werte, allerdings mit einer eingeschränkten Ordinate (98% bis 100%), um die unterschiedlichen Ergebnisse für die Einzel-Ontologien zu verdeutlichen.

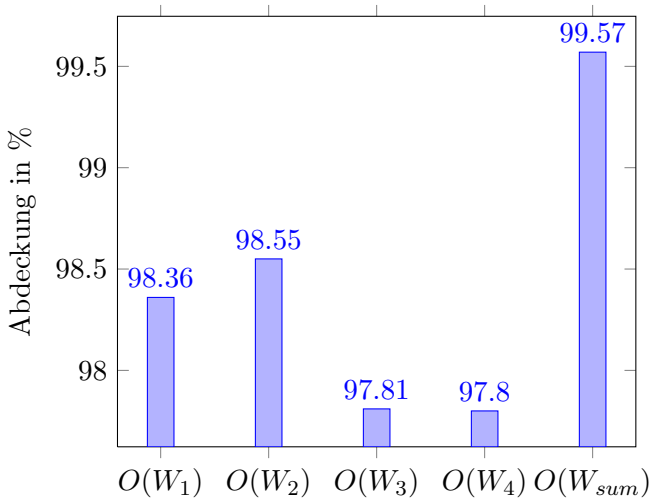

Abbildung 6.3.: Relative Synonym-Abdeckung (Ordinate 98% bis 100%)

Insgesamt sind diese Ergebnisse positiv zu bewerten. Besonders die Erkenntnis, dass der Fehler (Individuen, denen kein Synset zugewiesen werden kann) nicht linear mit der Anzahl der Individuen wächst, ist für das weitere Gesamt-Projekt von entscheidender Bedeutung. Schließlich soll später ein möglichst großer Teil der *Alice*-Objekte modelliert werden, was zu sehr großen Ontologien führt. Dass die Abdeckung mit der Größe der Ontologie zunimmt, respektive der relative Fehler abnimmt, stellt somit einen großen Erfolg dar.

Die Zunahme der Synonym-Abdeckung kann damit begründet werden, dass die meisten Individuen, die kein Synset erhalten, einer bestimmten Problemklasse angehören: Individuen, der Ontologie-Klasse „AliceQuestions“ enthalten mitunter Textfelder[1], die nicht weiter verarbeitet werden können. So lautet für das Individuum `edu.cmu.cs.stage3.alice.core.question.NumberIsEqualTo` das zugehörige Textfeld „<a>==<b>“. WordNet ist es aber nicht möglich, für „==“ ein entsprechenden Synset zu bestimmen. Diese Art von Fehler macht einen Großteil der nicht abgedeckten Individuen aus.

Innerhalb der Ontologie $O(W_1)$ sind beispielsweise alle sechs Individuen, die kein Synset erhalten, von dieser Art. Über dies hinaus konnte beobachtet werden, dass die beschriebenen Individuen in jeder der Stichproben-Ontologien vorkamen. Jene Feststellung erklärt zum einen die nahezu gleiche Abdeckung der Einzel-Welten als auch die Abdeckungssteigerung für Ontologie $O(W_{sum})$. Folglich kann angenommen werden, dass, zumindest für diese Fehlerfälle, die Nichtabdeckung nicht linear mit der Größe der Ontologie wächst, da diese Art Individuen nahezu eine Konstante über alle aus *Alice* extrahierten Ontologien darstellen. Wichtig ist in diesem Zusammenhang zu bemerken, dass diese Fehlerklasse für das Gesamtprojekt nahezu unerheblich ist, da Individuen der Klasse „AliceQuestions“ ohnehin nur innerhalb der graphischen Oberfläche von *Alice* genutzt werden. Für die Umsetzung von natürlichsprachlichem Text direkt in ein *Alice*-Skript (ohne Nutzung der graphischen Oberfläche) sollte somit keine Beeinträchtigung erfolgen.

Natürlich gibt es auch andere Individuen, für die WordNet keine Entsprechungen kennt. In der Ontologie $O(W_{sum})$ befinden sich zum Beispiel die Individuen „PJ“, „bonzai“ und

[1] Zur Erinnerung: Für Individuen der Klasse „AliceQuestions“ wird nicht der Methoden-Name sondern das zugehörige erklärende Textfeld betrachtet (siehe Kapitel 5.1).

„Strecher“, die kein Synset erhalten. Diese Art von Fehler kann leider nicht ausgeschlossen werden und erklärt sich entweder durch ungebräuchliche Namensgebung seitens *Alice* („PJ“ und „Strecher“) oder durch die Unvollständigkeit von WordNet („bonzai“). Ebenso wächst diese Art von Fehler linear mit der Größe der Ontologie. Die vollständige Liste der Individuen der betrachteten Ontologien, für die kein Synset bestimmt werden konnte, befindet sich in Anhang C.

6.2. Korrektheit des relevanten Wortes

Zur Evaluation des gewählten relevanten Wortes (siehe Kapitel 4.5) wurde ausschließlich die Ontologie $O(W_{sum})$ verwendet. Da die Individuen der Ontologien $O(W_1)$ bis $O(W_4)$ in $O(W_{sum})$ enthalten sind, tritt für die Betrachtung des relevanten Wortes kein Informationsverlust auf. Unter den Individuen von $O(W_{sum})$ wurden zufällig 300 zur näheren Betrachtung ausgewählt. Wie auch in Tabelle 6.2 nachzuvollziehen ist, wird 264 dieser Individuen die Wortart „Substantiv“, 32 „Verb“ und lediglich vier „Adjektiv“ zugeordnet.

Tabelle 6.2.: Zugeordnete Wortart der Stichproben-Individuen

Wortart	# zugeordneter Individuen	Anteil an der Stichprobe
Substantiv	264	88,00%
Verb	32	10,67%
Adjektiv	4	1,33%

Die deutliche Substantiv-Prägung lässt sich dadurch erklären, dass eine Vielzahl der Individuen der Ontologie-Super-Klasse „AliceModel“ angehören. Diese Klasse verlangt, wie in Kapitel 4.2 erläutert, die Wortart „Substantiv“. Die Klassenzuordnung der 300 betrachteten Beispiel-Individuen lässt sich in Tabelle 6.3 verfolgen.

Tabelle 6.3.: Super-Klassen-Zugehörigkeit der Stichproben-Individuen

Super-Klasse	# Anzahl zugeordneter Individuen	Anteil an der Stichprobe
AliceModel	254	84,67%
UserDefinedMethod	28	9,33%
Pose	8	2,67%
AliceQuestion	8	2,67%
AliceMethod	2	0,67%

Für diese 300 Individuen wurde experimentell untersucht, ob das bestimmte relevante Wort der Wortgruppe korrekt gewählt wurde. Hierbei sind drei unterschiedliche Ergebnisse möglich: Besteht die Wortgruppe nur aus einem Wort, wird die Wahl als „korrekt (nur ein Wort)“ eingestuft. Besteht die Wortgruppe hingegen aus mehreren Wörtern und die Wahl war ebenfalls richtig, wird als Ergebnis „korrekt (Auswahl)“ gewählt. Die Einstufung als korrekt unterliegt zum einen den in Kapitel 4.5 formulierten Anforderungen an das relevante Wort (Verben für Methoden, zweites Substantiv für Modelle usw.). Zum anderen wurde überprüft, ob diese Wahl auch für das konkrete Individuum zutrifft. Ist dies nicht der Fall, so wird es als „nicht eindeutig“ eingestuft. Trifft keine der zuvor genannten Möglichkeiten zu, so wird als Ergebnis „falsch“ gewählt. Das Diagramm in Abbildung 6.4 zeigt die entstandenen Ergebnisse. Tabelle 6.4 gibt zusätzlich die relative Häufigkeit an.

Dass der große Teil der ausgewählten Wörter als „korrekt“ eingestuft werden konnte, lässt sich hauptsächlich auf die Anwendbarkeit des trivialen Falls, bei dem keine Wahl erfolgen muss, zurückführen. Viele weitere korrekt gewählte relevante Wörter können mithilfe der Definition von Miller [Mil11] begründet werden (siehe Kapitel 4.5), die besagt, dass das

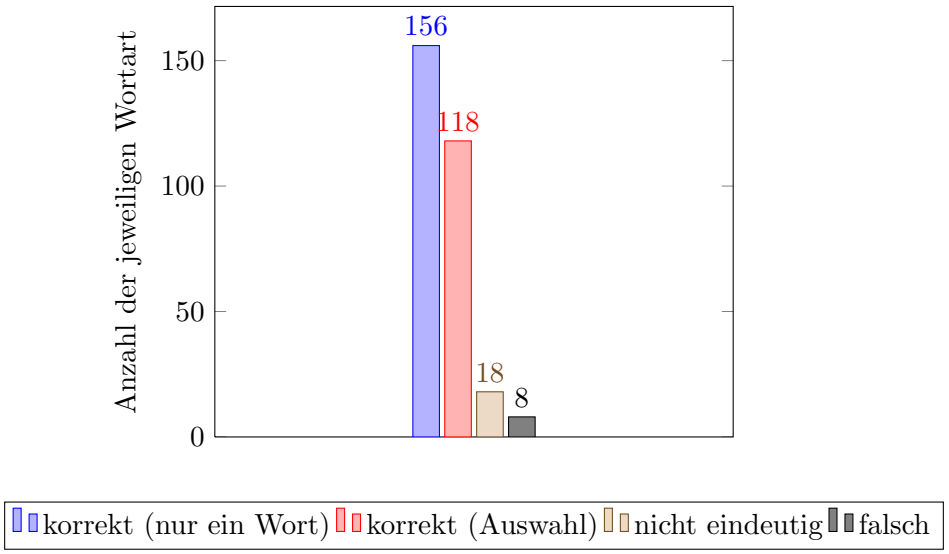

Abbildung 6.4.: Korrektheit des relevanten Wortes

Tabelle 6.4.: Bewertung der Wahl für das relevante Wort der Stichproben-Individuen

Wertung	# Individuen	Anteil an der Stichprobe
korrekt (nur ein Wort)	156	52,00%
korrekt (Auswahl)	118	39,33%
Summe (korrekt)	274	91,33%
nicht eindeutig	18	6,00%
falsch	8	2,67%
Summe (falsch)	26	8,66%

wichtigste Wort einer Wortgruppe jenes ist, welches essentiell ist und durch die anderen Wörter lediglich modifiziert wird. Eben dies gilt für Individuen wie `leftLeg` (relevantes Wort: „leg“) oder `Left_thigh` (relevantes Wort: „thigh“). Derartige Wortgruppen machen einen Großteil der Namen innerhalb von *Alice* aus.

Zuletzt sollen je zwei charakteristische Beispiele für nicht eindeutige und falsche Zuordnung diskutiert werden. Typische für eine nicht eindeutig mögliche Auswahl ist das Individuum `ArcticLandscape_AlaskanMountains`. Egal welche Maßstäbe zugrunde gelegt werden, eine eindeutige Auswahl für ein sinngebendes Wort ist an dieser Stelle (selbst für einen Menschen) unmöglich. In diesem Fall wurde das Wort „Landscape“ gewählt; eine Wahl die zumindest nicht falsch oder sinnentstellend ist. Als weiteres Beispiel sei das Individuum `body-rear` angegeben, die Wahl fiel hier auf „body“. Auch hier lässt sich nicht bestimmen, welches Wort für den inneren Zusammenhang wichtiger ist.

Eindeutig falsch hingegen wurde für das Individuum `gate_open` gewählt. Statt des essentiellen Verbs „open“ wurde hier „gate“ gewählt. Diese ungünstige Entscheidung ist dadurch zu erklären, dass das Wort „gate“ (Tor) neben der gebräuchlichen Substantiv-Bedeutung zusätzlich eine Verbform im Sinne von „versperren, ausblenden“ besitzt. Jene Mehrdeutigkeit kommt jedoch nur zum tragen, da der POS-Tagger das erste Wort der Wortgruppe fälschlich als Verb markiert hat. Diese Art Fehler treten häufiger auf, so auch im Fall des Individuums `HeadNod` (gewählt: „nod“). Tatsächlich lassen sich alle detektierten Fehler in der Wahl des relevanten Wortes auf ungenaue Markierungen des POS-Taggers zurückführen. Alle sechs falsch ausgewählten relevanten Wörter der Stichprobe sind in Tabelle 6.5 zusammen mit der tatsächlich richtigen Wahl verzeichnet.

Tabelle 6.5.: Falsch gewählte relevante Wörter der Stichproben-Individuen

Individuum	gesuchte Wortart	falsches Wort	korrekte Wahl
`concert_stage~~treble Right`	Substantiv	right	treble
`zombie~~zombie1_full_ come_out`	Verb	zombie	come
`gate~~gate_open`	Verb	gate	open
`Scientist_woman~~torso ~~HelmetRight`	Substantiv	right	helmet
`Penguin1~~wing_flap`	Verb	wing	flap
`Husky~~HeadNod`	Verb	head	nod

6.3. Synonym-Reduktion

In Kapitel 4.9 wurde ein Vorgehen beschrieben, um unter allen zuvor gefunden Synsets, diejenigen auszuwählen, die tatsächlich in den Kontext passen. Nachstehend (siehe Diagramm in Abbildung 6.5) sind die Ergebnisse der empirischen Überprüfung zu sehen.

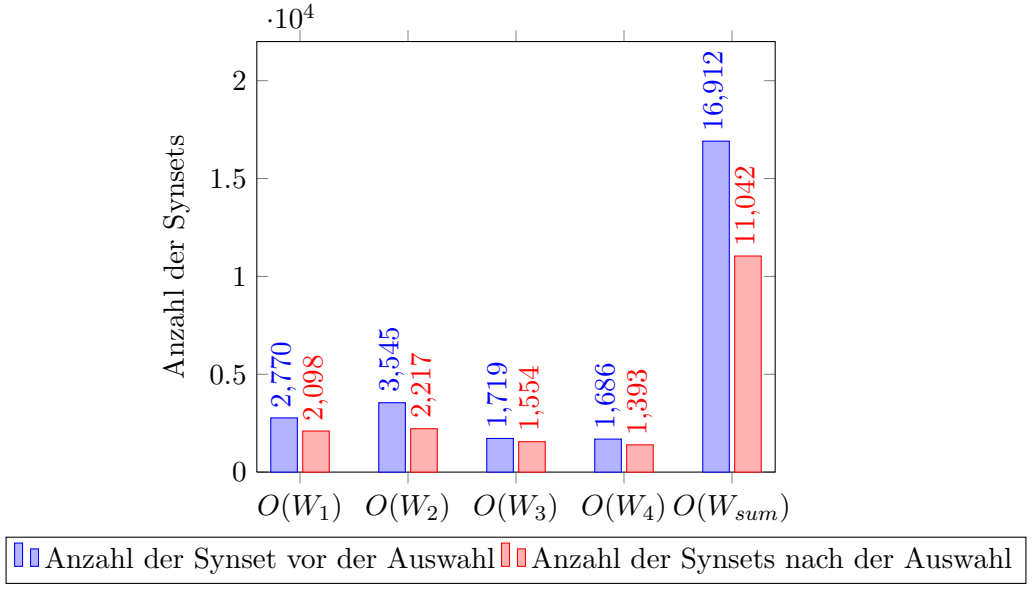

Abbildung 6.5.: Synonym-Auswahl – Anzahl der Synsets

Für alle Ontologien konnten Synsets aus der ursprünglichen Menge ausgeschlossen und diese folglich reduziert werden. Das Verfahren funktioniert somit grundsätzlich. Allerdings schwanken die Ergebnisse stark. Dies wird besonders deutlich, wird das Diagramm in Abbildung 6.6 zur Betrachtung hinzugezogen. Die Punkte im Diagramm verdeutlichen hierbei die absolute Reduktion, die Balken zusätzlich die relative.

Dass die absoluten Reduktionswerte derart unterschiedlich sind, erklärt sich über die unterschiedliche Größe der Ontologien. Sind zu Beginn mehr Synsets vorhanden, können natürlich auch mehr aussortiert werden. Aus diesem Grund erzielt die Ontologie $O(W_{sum})$ die höchsten Absolutwerte. Doch sowohl Ontologie $O(W_1)$ und $O(W_2)$ als auch $O(W_3)$ und $O(W_4)$ haben in etwa eine vergleichbare Größe (siehe Tabelle 6.1) als auch eine ähnliche Anzahl Synsets vor der Reduktion (siehe Diagramm in Abbildung 6.5); trotzdem sind die erzielten Ergebnisse der Synset-Reduktion sehr unterschiedlich. Besonders die Relativwerte sollten erwartungsgemäß gleichmäßigere Ergebnise vorweisen. Tatsächlich schwanken aber die Werte zwischen 37,46% für $O(W_2)$ und 9,60% für $O(W_3)$. Für diesen

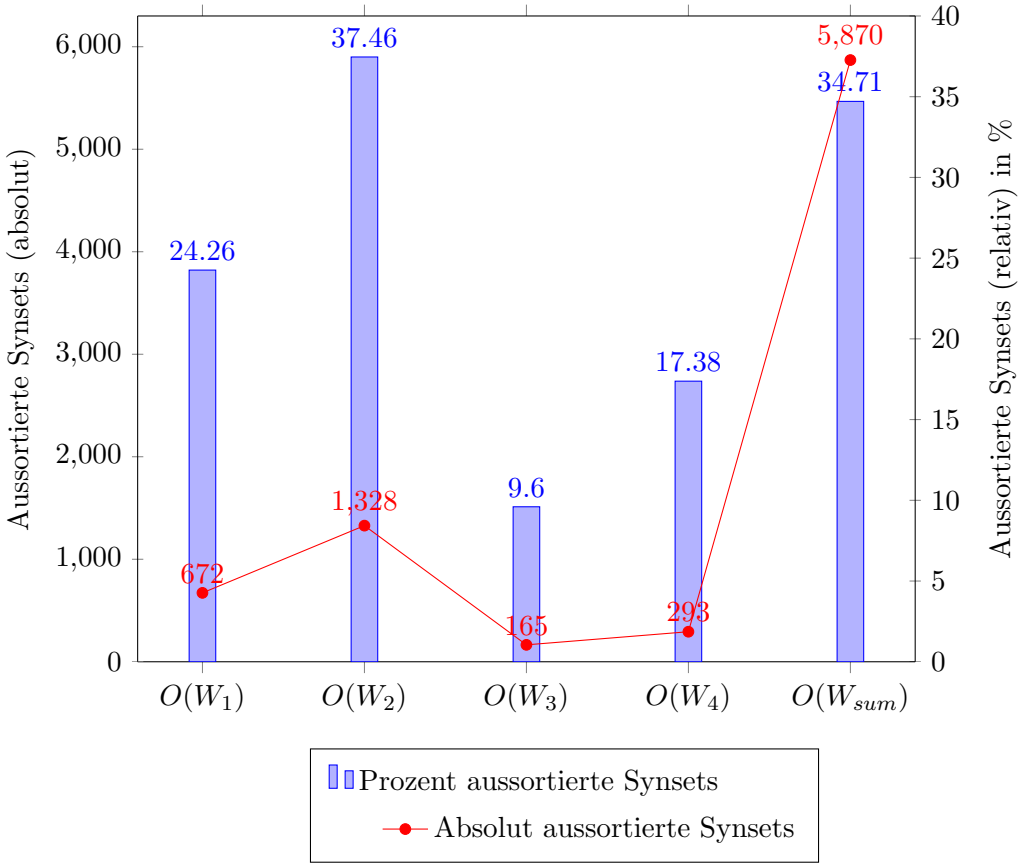

Abbildung 6.6.: Synonym-Auswahl – absolute und relative Reduktion

Effekt gibt es zwei mögliche Erklärungen: Zum einen werden nur Synsets, die zu Individuen der Super-Klasse „AliceModel“ gehören, einer Analyse unterzogen und gegebenenfalls aussortiert (siehe Kapitel 5.7). Schwanken innerhalb der Ontologien die Anzahl der Individuen dieser Klasse, ändert sich auch das Reduktionsergebnis. Zum anderen hängt die Entscheidung, ob ein Synset in den Kontext passt, zum größten Teil von dessen Einbettung in WordNet ab. Nicht jedes Synset besitzt in WordNet Meronyme, Holonyme oder Hyperonyme, zum einen weil tatsächlich keine existieren, zum anderen weil WordNet keine vollständige Abbildung der Wirklichkeit ist. Folglich fehlen Beziehungen, die für die Synset-Auswahl benutzt werden könnten. Die Performanz des Verfahrens hängt somit stark von den betrachteten Synsets ab. Dies Kombination der beiden Effekte erklärt die starken Schwankungen der Reduktionswerte innerhalb der Stichprobe.

Es kann jedoch beobachtet werden, dass größere Ontologien bessere relative Reduktionsergebnisse liefern. $O(W_3)$ ist die Ontologie mit der geringsten Individuenzahl, $O(W_2)$ die mit der höchsten, der Ontologien der Einzel-Welten. Ebenso liefert $O(W_{sum})$ mit 34,71% ein gutes Reduktionsergebnis. Eine mögliche Erklärung hierfür ist folgende: Große Ontologien haben erwartungsgemäß mehr Individuen der Art `head`, `body`, `leftEye` usw.. Diesen Individuen werden zunächst durch WordNet sehr viele Synsets zugeordnet. Ebenso können aber eine Vielzahl dieser Synsets aufgrund der hierarchischen Strukturen sehr effektiv aussortiert werden (siehe Kapitel 4.9 und 5.7).

Die Korrektheit der Auswahl soll an dieser Stelle, aufgrund des erheblichen Aufwands, nicht erschöpfend untersucht werden. Es sei jedoch darauf hingewiesen, dass das in Kapitel 4.9 beschriebene Verfahren konservativ reduziert. Das bedeutet, es werden eher Synsets in der Menge belassen als aussortiert. Nur wenn eindeutig treffendere Synsets existieren, wird ein anderes eliminiert. Die Wahrscheinlichkeit eines Informationsverlustes ist somit

gering. Diese These sei anhand zweier Ontologie-Individuen exemplarisch belegt:

- `EskimoGirl~~Head`: Vor der Auswahl besitzt dieses Individuum 32 Synsets, unter anderen: Noun@5469706[head, caput]; Noun@10318027[principal, school principal, head teacher, head]; Noun@7271830[headway, head]; Noun@6208798[head, head word].

  Nach der Auswahl bleibt lediglich dass Synset *Noun@5469706[head, caput]* übrig, welches das einzig sinnvolle im Kontext „EskimoGirl“ ist.

- `Igloo~~IglooDoor`: Diesem Individuum werden folgende Synsets zugewiesen: Noun@3188473[door]; Noun@3190785[doorway, door, room access, threshold]; Noun@5119817[door]; Noun@3189071[door]; Noun@3188929[door].

  Die Synset-Auswahl bleibt in diesem Fall erfolglos, da über WordNet kein Zusammenhang zwischen den Begriffen „Igloo“ (Iglu) und „door“ (Tür) hergestellt werden kann. Folglich bleiben alle Synsets erhalten.

Gleichzeitig konnte bei der Betrachtung von 100 zufällig gewählten Individuen kein Fall gefunden werden, bei dem ein Synset aussortiert wurde, obwohl es in dem vorgegebenen Kontext gepasst hätte.

7. Zusammenfassung

Natürlichsprachliche Programmierung ist eine interessante Möglichkeit, Personen mit wenig oder keiner Programmiererfahrung dieses vielfältige Betätigungsfeld zu erschließen. Gleichzeitig wird Programmieren so zu einer kreativen und schöpferischen Tätigkeit.

Das Projekt „Natürlichsprachliche Programmierung“ verfolgt hierbei einen völlig neuen Ansatz, wird doch die Programmiersprache *Alice* anstelle des natürlichsprachlichen Textes modelliert. Dies erhöht die Wahrscheinlichkeit, einen Text erfolgreich zunächst auf dieses Modell in Form einer Ontologie abzubilden und später in Quelltext umzusetzen.

Die Teilaufgabe, die in der vorliegenden Arbeit gelöst wurde, befasst sich mit der lexikographischen Erweiterung der Ontologie-Elemente, welche die *Alice*-Objekte und -Methoden modellieren. Dies vereinfacht später die Assoziation von Text-Korpus und Ontologie. Die Arbeit stützt sich dabei auf die Annahme, dass in *Alice* verwendete Objekte und Methoden sogenannte „sprechende“ Namen tragen, das bedeutet, die Namen geben Auskunft über die Funktionalität. Um die Namen lexikographisch untersuchen zu können, werden diese zunächst in ein natürlichsprachliches Format übertragen. Es entstehen kurze Wortgruppen, deren sinngebendes Wort mittels WordNet mit einer Synonymmenge verknüpft wird. Natürlich führt dieses Vorgehen zu einer großen Synonymmenge, die anschließend mit unterschiedlichen Verfahren reduziert wird. Die gesammelten Information werden innerhalb der Ontologie als Annotation gespeichert.

Im Zuge der Evaluation konnte gezeigt werden, dass die vorgestellte Arbeit eine Synonym-Abdeckung von ca. 98%, für entsprechend große Ontologien von über 99%, erreicht. Das heißt, nahezu jedes Ontologie-Individuum erhält mindestens ein Synonym. Ebenso konnte gezeigt werden, dass die Wahl des sinngebenden Wortes (head-of-phrase) in über 91% der Fälle korrekt war. Die Synonym-Auswahl erreicht relative Reduktionen im Bereich von 9% bis 38%, wobei die Schwankungen zum größten Teil auf die Struktur von WordNet beziehungsweise dessen Unvollständigkeit zurückgeführt werden konnten.

Zukünftig kann insbesondere die Synonym-Auswahl verbessert werden. Bis hierher wurde das Hauptaugenmerk darauf gelegt, keinesfalls Informationen zu verlieren. Das Verfahren könnte aggressiver abgestimmt werden, so dass mehr Synonyme aussortiert würden. Hierzu wäre die Einflechtung von weiterem Hintergrundwissen, neben WordNet, zweckdienlich; denkbar wäre vor allem die Nutzung einer Weltwissen-Ontologie. Die Auswahl des relevanten Wortes kann ebenso verfeinert werden. Allerdings wird hierzu entweder ein verbesserter Wortart-Markierer benötigt, da alle momentan verursachten Fehler auf diesen zurückzuführen sind, oder ein anderes Verfahren wird konzipiert.

Da die Synonym-Erweiterung einer Ontologie in dieser Art neu ist, können die erzielten Ergebnissen nicht mit denen anderer Arbeiten verglichen werden. Somit kann erst mit Fortschreiten des Projekts „Programmieren in natürlicher Sprache“ abschließend geklärt werden, inwieweit die hier geleistete Arbeit tatsächlich erfolgreich war. Die Hauptanforderung der Auffindung von Synonymen zu sprechenden Objekt- und Methoden-Namen, konnte indes erfüllt werden.

Mithilfe der Erweiterungen der Ontologie kann im nächsten Projektschritt ein gegebener Textkorpus mit der Ontologie verknüpft werden. Hierbei helfen die Synonyme um Variationen des natürlichsprachlichen Textes auszugleichen. Wurden entsprechende Verknüpfung hergestellt, kann das eigentliche Ziel des Projektes „Programmieren in natürlicher Sprache“ verwirklicht werden: Die Umsetzung eines natürlichsprachlichen Textes in ein *Alice*-Skript.

Literaturverzeichnis

[BB79] Bruce W. Ballard und Alan W. Biermann: *Programming in natural language: “NLC” as a prototype.* In: *Proceedings of the 1979 annual conference*, ACM '79, Seiten 228–237, New York, NY, USA, 1979. ACM, ISBN 0-89791-008-7. http://doi.acm.org/10.1145/800177.810072.

[BD04] Bernd Brügge und Allen H. Dutoit: *Objektorientierte Softwaretechnik - mit UML, Entwurfsmustern und Java.* Pearson Studium, 2004, ISBN 978-3-8273-7082-2.

[CAB+00] Matthew Conway, Steve Audia, Tommy Burnette, Dennis Cosgrove und Kevin Christiansen: *Alice: lessons learned from building a 3D system for novices.* In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '00, Seiten 486–493, New York, NY, USA, 2000. ACM, ISBN 1-58113-216-6. http://alice.org/publications/chialice.pdf.

[Cho95] Noam Chomsky: *The Minimalist Program.* MIT Press, Cambridge, 1995.

[Con97] Matthew J. Conway: *Alice: Easy-to-Learn 3D Scripting for Novices.* Dissertation, Faculty of the School of Engineering and Applied Science, University of Virginia, Dezember 1997. http://alice.org/publications/ConwayDissertation.PDF.

[EHPVS09] E. Enslen, E. Hill, L. Pollock und K. Vijay-Shanker: *Mining source code to automatically split identifiers for software analysis.* In: *Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on*, Seiten 71–80, Mai 2009.

[GOS09] Nicola Guarino, Daniel Oberle und Steffen Staab: *What Is an Ontology?* In: Steffen Staab und Dr. Rudi Studer (Herausgeber): *Handbook on Ontologies*, International Handbooks on Information Systems, Seiten 1–17. Springer Berlin Heidelberg, 2009, ISBN 978-3-540-92673-3. http://dx.doi.org/10.1007/978-3-540-92673-3_0.

[GS03] Fausto Giunchiglia und Pavel Shvaiko: *Semantic matching.* The Knowledge Engineering Review, 18(03):265–280, 2003. http://dx.doi.org/10.1017/S0269888904000074.

[GSY04] Fausto Giunchiglia, Pavel Shvaiko und Mikalai Yatskevich: *S-Match: an Algorithm and an Implementation of Semantic Matching.* In: Christoph Bussler, John Davies, Dieter Fensel und Rudi Studer (Herausgeber): *The Semantic Web: Research and Applications*, Band 3053 der Reihe *Lecture Notes in Computer Science*, Seiten 61–75. Springer Berlin / Heidelberg, 2004, ISBN 978-3-540-21999-6. http://dx.doi.org/10.1007/978-3-540-25956-5_5.

[Hor08] Ian Horrocks: *Ontologies and the Semantic Web.* Communications of the ACM, 51(12):58 – 67, 2008, ISSN 00010782. http:

//www.redi-bw.de/db/ebsco.php/search.ebscohost.com/login.aspx?direct=true&db=buh&AN=35609280&site=ehost-live.

[KM06] Roman Knöll und Mira Mezini: *Pegasus: first steps toward a naturalistic programming language.* In: *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, OOPSLA '06, Seiten 542–559, New York, NY, USA, 2006. ACM, ISBN 1-59593-491-X. http://doi.acm.org/10.1145/1176617.1176628.

[LL05] Hugo Liu und Henry Lieberman: *Metafor: visualizing stories as code.* In: *Proceedings of the 10th international conference on Intelligent user interfaces*, IUI '05, Seiten 305–307, New York, NY, USA, 2005. ACM, ISBN 1-58113-894-6. http://doi.acm.org/10.1145/1040830.1040908.

[LLYM04] Shuang Liu, Fang Liu, Clement Yu und Weiyi Meng: *An effective approach to document retrieval via utilizing WordNet and recognizing phrases.* In: *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '04, Seiten 266–272, New York, NY, USA, 2004. ACM, ISBN 1-58113-881-4. http://doi.acm.org/10.1145/1008992.1009039.

[LWSC08] Wei Liu, Albert Weichselbraun, Arno Scharl und Elizabeth Chang: *Semi-Automatic Ontology Extension Using Spreading Activation.* 2008. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.101.3677.

[Man03] Hinrich Manning, Christopher D. ; Schütze: *Foundations of statistical natural language processing.* MIT Press, Cambridge, Mass. [u.a.], 6. print. with corr. Auflage, 2003, ISBN 0-262-13360-1 ; 978-0-262-13360-9. http://www.ulb.tu-darmstadt.de/tocs/121246302.pdf.

[Mil95] George A. Miller: *WordNet: a lexical database for English.* Commun. ACM, 38(11):39–41, November 1995, ISSN 0001-0782. http://doi.acm.org/10.1145/219717.219748.

[Mil11] J. Miller: *A Critical Introduction to Syntax.* Continuum Critical Introductions to Linguistics. Bloomsbury, 2011, ISBN 9780826497048. http://books.google.com/books?id=aOUQHXUzR9AC.

[MSM93] Mitchell P. Marcus, Beatrice Santorini und Mary Ann Marcinkiewicz: *Building a Large Annotated Corpus of English: The Penn Treebank.* Computational Linguistics, 19(2):313–330, Juni 1993.

[Noy09] Natalya F. Noy: *Ontology Mapping.* In: Steffen Staab und Dr. Rudi Studer (Herausgeber): *Handbook on Ontologies*, International Handbooks on Information Systems, Seiten 573–590. Springer Berlin Heidelberg, 2009, ISBN 978-3-540-92673-3. http://dx.doi.org/10.1007/978-3-540-92673-3_26.

[Pet12] Oleg Peters: *Programmieren in natürlicher Sprache: Aufbau einer Alice-Ontologie.* Bachelor's Thesis, Karlsruher Institut für Technologie (KIT) – IPD Tichy, 2012. http://www.ipd.kit.edu/Tichy/theses.php?id=202.

[Pla03] I. Plag: *Word-Formation in English.* Cambridge Textbooks in Linguistics. Cambridge University Press, 2003, ISBN 9780521525633. http://books.google.de/books?id=78KFCIHtJS4C.

[PPM04] Ted Pedersen, Siddharth Patwardhan und Jason Michelizzi: *WordNet::Similarity: measuring the relatedness of concepts.* In: *Demonstration*

Papers at HLT-NAACL 2004, HLT-NAACL–Demonstrations '04, Seiten 38–41, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics. http://dl.acm.org/citation.cfm?id=1614025.1614037.

[PRZH00] David Price, Ellen Rilofff, Joseph Zachary und Brandon Harvey: *NaturalJava: a natural language interface for programming in Java*. In: *Proceedings of the 5th international conference on Intelligent user interfaces*, IUI '00, Seiten 207–211, New York, NY, USA, 2000. ACM, ISBN 1-58113-134-8. http://doi.acm.org/10.1145/325737.325845.

[TKMS03] Kristina Toutanova, Dan Klein, Christopher D. Manning und Yoram Singer: *Feature-rich part-of-speech tagging with a cyclic dependency network*. In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, Seiten 173–180, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. http://dx.doi.org/10.3115/1073445.1073478.

[TM00] Kristina Toutanova und Christopher D. Manning: *Enriching the knowledge sources used in a maximum entropy part-of-speech tagger*. In: *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics - Volume 13*, EMNLP '00, Seiten 63–70, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics. http://dx.doi.org/10.3115/1117794.1117802.

Anhang

A. Das Penn-Tagset

Das Penn-Tagset mit den POS-Tags CC bis WRB sowie den zwölf Sonderkennzeichnungen für Zeichen und Symbole:

CC	Coordinating conjunction	TO	*to*
CD	Cardinal number	UH	Interjection
DT	Determiner	VB	Verb, base form
EX	Existential *there*	VBD	Verb, past tense
FW	Foreign word	VBG	Verb, gerund or present participle
IN	Preposition or subordinating conjunction	VBN	Verb, past participle
JJ	Adjective	VBP	Verb, non-3rd person singular present
JJR	Adjective, comparative	VBZ	Verb, 3rd person singular present
JJS	Adjective, superlative	WDT	*wh*-determiner
LS	List item marker	WP	*wh*-pronoun
MD	Modal	WP$	Possessive *wh*-pronoun
NN	Noun, singular or mass	WRB	*Wh*-adverb
NNS	Noun, plural	#	Pound sign
NP	Proper noun, singular	$	Dollar sign
NPS	Proper noun, plural	.	Sentence-final punctuation
PDT	Predeterminer	,	Comma
POS	Possessive ending	:	Colon, semi-colon
PP	Personal pronoun	(	Left bracket character
PP$	Possessive pronoun	)	Right bracket character
RB	Adverb	"	Straight double quote
RBR	Adverb, comparative	‘	Left open single quote
RBS	Adverb, superlative	“	Left open double quote
RP	Particle	’	Right close single quote
SYM	Symbol	”	Right close double quote

Tabelle A.1.: Das Penn-Tagset nach Marcus et al. [MSM93].

B. Die Klassenhierarchie der *Alice*-Ontologien

Die Liste aller verwendeten Klassen innerhalb der *Alice*-Ontologien, aus der Arbeit von Oleg Peters [Pet12]:

`AliceModel`

Als Gegenstück zu den Modellen in den Alice-Welten vereinigt die Klasse `AliceModel` sowohl in der Szenerie sichtbare Objekte als auch vom Rahmenwerk vordefinierte Objekte:

- `World` enthält alle Szenerien, die extrahiert in der Ontologie enthalten sind;
- `FirstClassModel` enthält alle eigenständigen Objekte der Szenerien;
- `Model` umfasst alle übrigen 3D-Objekte, aus welchen FirstClass-Objekte zusammengesetzt sind;
- `Geometry` enthält die geometrischen Primitive der Szenerien;
- `Camera` enthält die verwendeten Kamera-Objekte;
- und `Light` umfasst alle in den Alice-Welten verwendeten Lichttypen.

`Pose`

Die in den Alice-Objekten definierten Posen bildet der Extraktor auf die Instanzen der Klasse `Pose` ab.

`Method`

In dem Konzept Method fassen wir alle Methoden und Funktionen zusammen, die in den Welten verfügbar sind. Dazu unterscheiden wir genauso wie im Rahmenwerk zwischen vordefinierten und benutzerdefinierten Funktionen und Methoden. Wir führen folgende Klassen ein:

- `AliceMethod`, die alle im Rahmenwerk definierten Methoden als Instanzen enthält;
- `AliceQuestion` für die im Rahmenwerk verfügbaren Funktionen;
- `UserDefinedMethod`, auf die wir die benutzerdefinierten Methoden abbilden und
- `UserDefinedQuestion`, in der wir die in den Welten gefundenen Funktionen speichern.

`Property`

Die übergabeparameter für die Methoden und Parameter der Alice-Objekte werden als Instanzen der Klasse `Property` verwaltet. Dabei tragen die Instanzen die Namen der zugehörigen Parameter.

`DataType`

Um die Typsicherheit im Aufbau der Alice-Welten im Gesamtprojekt zu gewährleisten, führen wir die Klasse `DataType` ein, deren Individuen die Datentypen repräsentieren.

C. Liste der Individuen die kein Synset erhalten

Die Tabelle listet alle Individuen einer jeden innerhalb der Evaluation betrachteten Ontologie. Individuen die mit „edu.[...].[...]“ abgekürzt sind lauten tatsächlich wie folgt: „edu.cmu.cs.stage3.alice.core.question.[...]“.

Ontologie	nicht abgedecktes Individuum	Textfeld (soweit vorhanden)
$O(W_1)$	edu.[...].NumberIsGreaterThanOrEqualTo	<a>>=<b>
	edu.[...].NumberIsLessThan	<a><<b>
	edu.[...].NumberIsEqualTo	<a>==<b>
	edu.[...].NumberIsNotEqualTo	<a>!=<b>
	edu.[...].NumberIsLessThanOrEqualTo	<a><=<b>
	edu.[...].RightUpForward	<right>, <up>, <forward>
$O(W_2)$	edu.[...].NumberIsGreaterThanOrEqualTo	<a>>=<b>
	edu.[...].NumberIsLessThan	<a><<b>
	edu.[...].NumberIsEqualTo	<a>==<b>
	edu.[...].NumberIsNotEqualTo	<a>!=<b>
	edu.[...].NumberIsLessThanOrEqualTo	<a><=<b>
	edu.[...].RightUpForward	<right>, <up>, <forward>
	PJ	
$O(W_3)$	edu.[...].NumberIsGreaterThanOrEqualTo	<a>>=<b>
	edu.[...].NumberIsLessThan	<a><<b>
	edu.[...].NumberIsEqualTo	<a>==<b>
	edu.[...].NumberIsNotEqualTo	<a>!=<b>
	edu.[...].NumberIsLessThanOrEqualTo	<a><=<b>
	edu.[...].RightUpForward	<right>, <up>, <forward>
$O(W_4)$	edu.[...].NumberIsGreaterThanOrEqualTo	<a>>=<b>
	edu.[...].NumberIsLessThan	<a><<b>
	edu.[...].NumberIsEqualTo	<a>==<b>
	edu.[...].NumberIsNotEqualTo	<a>!=<b>
	edu.[...].NumberIsLessThanOrEqualTo	<a><=<b>
	edu.[...].RightUpForward	<right>, <up>, <forward>
$O(W_{sum})$	edu.[...].NumberIsGreaterThanOrEqualTo	<a>>=<b>
	edu.[...].NumberIsLessThan	<a><<b>
	edu.[...].NumberIsEqualTo	<a>==<b>
	edu.[...].NumberIsNotEqualTo	<a>!=<b>
	edu.[...].NumberIsLessThanOrEqualTo	<a><=<b>
	edu.[...].RightUpForward	<right>, <up>, <forward>
	bonzai1	
	bonzai	
	PJ	
	Strecher	